MouseDB Documentation

Release 0.2.1dev

Dave Bridges, Ph.D.

March 26, 2011

CONTENTS

1 MouseDB Concepts **3**
- 1.1 Animal Module . . . 3
- 1.2 Data Module . . . 4
- 1.3 Timed Matings Module . . . 4
- 1.4 Groups Module . . . 4

2 MouseDB Installation **5**
- 2.1 Configuration . . . 5
- 2.2 Software Dependencies . . . 5
- 2.3 Installation . . . 6
- 2.4 Database Setup . . . 6
- 2.5 Web Server Setup . . . 6
- 2.6 Final Configuration and User Setup . . . 7
- 2.7 Testing . . . 7

3 Animal Data Entry **9**
- 3.1 Newborn Mice or Newly Weaned Mice . . . 9
- 3.2 Newborn Mice . . . 9
- 3.3 Weaning Mice . . . 9
- 3.4 Cage Changes (Not Weaning) . . . 9
- 3.5 Genotyping or Ear Tagging . . . 10
- 3.6 Marking Mice as Dead . . . 10

4 Studies and Experimental Setup **11**

5 Measurement Entry **13**
- 5.1 Studies . . . 13
- 5.2 Experiment Details . . . 13
- 5.3 Measurements . . . 13

6 Automated Documentation **15**
- 6.1 Data Package . . . 15
- 6.2 Animals Package . . . 22
- 6.3 Timed Mating Package . . . 29
- 6.4 Groups Package . . . 32

7 Indices and tables **35**

Python Module Index **37**

Index 39

Contents:

CHAPTER

ONE

MOUSEDB CONCEPTS

Data storage for MouseDB is separated into packages which contain information about animals, and information collected about animals. There is also a separate module for timed matings of animals. This document will describe the basics of how data is stored in each of these modules.

1.1 Animal Module

Animals are tracked as individual entities, and given associations to breeding cages to follow ancestry, and strains.

1.1.1 Animal

Most parameters about an animal are set within the animal object. Here is where the animals strain, breeding, parentage and many other parameters are included. Animals have foreignkey relationships with both Strain and Breeding, so an animal may only belong to one of each of those. As an example, a mouse cannot come from more than one Breeding set, and cannot belong to more than one strain.

Backcrosses and Generations

For this software, optional tracking of backcrosses and generations is available and is stored as an attribute of an animal. When an inbred cross is made against a pure background, the backcross increases by 1. When a heterozygote cross is made, the generation increases by one. As an example, for every time a mouse in a C57/BL6 background is crossed against a wildtype C57/B6 mouse, the backcross (but not the generation) increases by one. For every time a mutant strain is crosses against itself (either vs a heterozygote or homozygote of that strain), the generation will increase by one. Backcrosses should typically be performed against a separate colony of purebred mouse, rather than against wild-type alleles of the mutant strain.

1.1.2 Breeding Cages

A breeding cage is defined as a set of one or more male and one or more female mice. Because of this, it is not always clear who the precise parentage of an animal is. If the parentage is known, then the Mother and Father fields can be set for a particular animal. In the case of Active, if an End field is specified, then the Active field is set to False. In the case of Cage, if a Cage is provided, and animals are specified under Male or Females for a Breeding object, then the Cage field for those animals is set to that of the breeding cage. The same is true for both Rack and Rack Position.

1.1.3 Strains

A strain is a set of mice with a similar genetics. Importantly strains are separated from Backgrounds. For example, one might have mice with the genotype ob/ob but these mice may be in either a C57-Black6 or a mixed background. This difference is set at the individual animal level. The result of this is that a query for a particular strain may then need to be filtered to a specific background.

1.2 Data Module

Data (or measurements) can be stored for any type of measurement. Conceptually, several pieces of data belong to an experiment (for example several mice are measured at some time) and several experiments belong to a study. Measurements can be stored independent of experiments and experiments can be performed outside of the context of a study. It is however, perfered that measurements are stored within an experiment and experiments are stored within studies as this will greatly facilitate the organization of the data.

1.2.1 Studies

In general studies are a collection of experiments. These can be grouped together on the basis of animals and/or treatment groups. A study must have at least one treatment group, which defines the animals and their conditions.

1.2.2 Experiments

An experiment is a collection of measurements for a given set of animals. In general, an experiment is defined as a number of measurements take in a given day.

1.2.3 Measurements

A measurement is an animal, an assay and a measurement value. It can be associated with an experiment, or can stand alone as an individual value. Measurements can be viewed in the context of a study, an experiment, a treatment group or an animal by going to the appropriate page.

1.3 Timed Matings Module

Timed matings are a specific type of breeding set. Generally, for these experiments a mating cage is set up and pregnancy is defined by a plug event. Based on this information, the age of an embryo can be estimated. When a breeding cage is defined, one option is to set this cage as a timed mating cage (ie Timed_Mating=True). If this is the case, then a plug event can be registered and recorded for this mating set. If the mother gives birth then this cage is implicitly set as a normal breeding cage.

1.4 Groups Module

This app defines generic Group and License information for a particular installation of MouseDB. Because every page on this site identifies both the Group and data restrictions, at a minimum, group information must be provided upon installation (see installation instructions).

CHAPTER

TWO

MOUSEDB INSTALLATION

2.1 Configuration

MouseDB requires both a database and a webserver to be set up. Ideally, the database should be hosted separately from the webserver and MouseDB installation, but this is not necessary, as both can be used from the same server. If you are using a remote server for the database, it is best to set up a user for this database that can only be accessed from the webserver. If you want to set up several installations (ie for different users or different laboratories), you need separate databases and MouseDB installations for each. You will also need to set up the webserver with different addresses for each installation.

2.2 Software Dependencies

1. **Python**. Requires Version 2.6, is not yet compatible with Python 3.0. Download from http://www.python.org/download/.
2. **MouseDB source code**. Download from one of the following:

1. Using **pip or easy_install**. If setuptools (available at http://pypi.python.org/pypi/setuptools) is installed type **pip install mousedb** at a command prompt.
2. http://github.com/davebridges/mousedb/downloads for the current release. If you will not be contributing to the code, download from here.
3. http://github.com/davebridges/mousedb for the source code via Git. If you might contribute code to the project use the source code.

Downloading and/or unzipping will create a directory named mousedb. You can update to the newest revision at any time either using git or downloading and re-installing the newer version. Changing or updating software versions will not alter any saved data, but you will have to update the localsettings.py file (described below).

3. **Database software**. Recommended to use mysql, available at http://dev.mysql.com/downloads/mysql/ . It is also possible to use SQLite, PostgreSQL, MySQL, or Oracle. See http://docs.djangoproject.com/en/1.2/topics/install/#database-installation for more information. You will also need the python bindings for your database. If using MySQL python-mysql will be installed below.
4. **Webserver**. Apache is recommended, available at http://www.apache.org/dyn/closer.cgi . It is also possible to use FastCGI, SCGI, or AJP. See http://docs.djangoproject.com/en/1.2/howto/deployment/ for more details. The recommended way to use Apache is to download and enable mod_wsgi. See http://code.google.com/p/modwsgi/ for more details.

2.3 Installation

1. Navigate into mousedb folder
2. Run **python setup.py install** to get dependencies. If you installed via pip, this step is not necessary (but wont hurt). This will install the dependencies South, mysql-python and django-ajax-selects.
3. Run **python bootstrap.py** to get the correct version of Django and to set up an isolated environment. This step may take a few minutes.
4. Run **bin\buildout** to generate django, test and wsgi scripts. This step may take a few minutes.

2.4 Database Setup

1. Create a new database. Check the documentation for your database software for the appropriate syntax for this step. You need to record the user, password, host and database name. If you are using SQLite this step is not required.
2. Go to mousedbsrcmousedblocalsettings_empty.py and edit the settings:

- ENGINE: Choose one of 'django.db.backends.postgresql_psycopg2','django.db.backends.postgresql', 'django.db.backends.mysql', 'django.db.backends.sqlite3', 'django.db.backends.oracle' depending on the database software used.
- NAME: database name
- USER: database user
- PASSWORD: database password
- HOST: database host

3. Save this file as **localsettings.py** in the same folder as localsettings_empty.py
4. Migrate into first mousedb directory and enter *django syncdb*. When prompted create a superuser (who will have all availabler permissions) and a password for this user.

2.5 Web Server Setup

You need to set up a server to serve both the django installation and saved files. For the saved files. I recommend using apache for both. The preferred setup is to use Apache2 with mod_wsgi. See http://code.google.com/p/modwsgi/wiki/InstallationInstructions for instructions on using mod_wsgi. The following is a httpd.conf example where the code is placed in **/usr/src/mousedb**:

```
Alias /robots.txt /usr/src/mousedb/src/mousedb/media/robots.txt
Alias /favicon.ico /usr/src/mousedb/src/mousedb/media/favicon.ico

Alias /mousedb-media/ /usr/src/mousedb/src/mousedb/media/
<Directory /usr/src/mousedb/src/mousedb/media>
     Order deny,allow
     Allow from all
</Directory>

Alias /static/ /usr/src/mousedb/src/mousedb/static/
<Directory /usr/src/mousedb/src/mousedb/static>
     Order deny,allow
     Allow from all
```

```
</Directory>

<Directory /usr/src/mousedb/bin>
     Order deny,allow
     Allow from all
</Directory>
WSGIScriptAlias /mousedb /usr/src/mousedb/bin/django.wsgi
```

If you want to restrict access to these files, change the Allow from all directive to specific domains or ip addresses (for example Allow from 192.168.0.0/99 would allow from 192.168.0.0 to 192.168.0.99)

2.6 Final Configuration and User Setup

- Go to *servername/mousedb/admin/groups/group/1* and name your research group and select a license if desired
- Go to *servername/mousedb/admin/auth/users/* and create users, selecting usernames, full names, password (or have the user set the password) and then choose group permissions.

2.7 Testing

From the mousedb directory run **bin\test** or **bin\django test** to run the test suite. See https://github.com/davebridges/mousedb/wiki/Known-Issues—Test-Suite for known issues. Report any additional errors at the issue page at https://github.com/davebridges/mousedb/issues.

CHAPTER

THREE

ANIMAL DATA ENTRY

3.1 Newborn Mice or Newly Weaned Mice

1. Go to Breeding Cages Tab
2. Click on Add/Wean Pups Button
3. Each row is a new animal. If you accidentaly enter an extra animal, check off the delete box then submit.
4. Leave extra lines blank if you have less than 10 mice to enter
5. If you need to enter more than 10 mice, enter the first ten and submit them. Go back and enter up to 10 more animals (10 more blank spaces will appear)

3.2 Newborn Mice

1. Enter Breeding Cage under Cage
2. Enter Strain
3. Enter Background (normally Mixed or C57BL/6-BA unless from the LY breeding cages in which case it is C57BL/6-LY5.2)
4. Enter Birthdate in format YYYY-MM-DD
5. Enter Generation and Backcross

3.3 Weaning Mice

1. If not previously entered, enter data as if newborn mice
2. Enter gender
3. Enter Wean Date in format YYYY-MM-DD
4. Enter new Cage number for Cage

3.4 Cage Changes (Not Weaning)

1. Find mouse either from animal list or strain list

2. Click the edit mouse button
3. Change the Cage, Rack and Rack Position as Necessary

3.5 Genotyping or Ear Tagging

1. Find mouse either from animal list or strain list, or through breeding cage
2. Click the edit mouse button or the Eartag/Genotype/Cage Change/Death Button
3. Enter the Ear Tag and/or select the Genotype from the Pull Down List

3.6 Marking Mice as Dead

3.6.1 Dead Mice (Single Mouse)

1. Find mouse from animal list or strain list
2. Click the edit mouse button
3. Enter the death date in format YYYY-MM-DD
4. Choose Cause of Death from Pull Down List

3.6.2 Dead Mice (Several Mice)

1. Find mice from breeding cages
2. Click the Eartag/Genotype/Cage Change/Death Button
3. Enter the death date in format YYYY-MM-DD
4. Choose the Cause of Death from Pull Down List

CHAPTER

FOUR

STUDIES AND EXPERIMENTAL SETUP

Set up a new study at /mousedb/admin/data/study/ selecting animals

You must put a description and select animals in one or more treatment groups

If you have more than 2 treatment groups save the first two, then two more empty slots will appear. For animals, click on the magnifying glass then find the animal in that treatment group and click on the MouseID. The number displayed now in that field will not be the MouseID, but don't worry its just a different number to describe the mouse. To add more animals, click on the magnifying glass again and select the next animal. There should be now two numbers, separated by commas in this field. Repeat to fill all your treatment groups. You must enter a diet and environment for each treatment. The other fields are optional, and should only be used if appropriate. Ensure for pharmaceutical, you include a saline treatment group.

CHAPTER

FIVE

MEASUREMENT ENTRY

5.1 Studies

If this measurement is part of a study (ie a group of experiments) then click on the plus sign beside the study field and enter in the details about the study and treatment groups. Unfortunately until i can figure out how to filter the treatment group animals in the admin interface, at each of the subsequent steps you will see all the animals in the database (soon hopefully it will only be the ones as part of the study group).

5.2 Experiment Details

- Pick experiment date, feeding state and resarchers
- Pick animals used in this experiment (the search box will filter results)
- Fasting state, time, injections, concentration, experimentID and notes are all optional

5.3 Measurements

- There is room to enter 14 measurements. If you need more rows, enter the first 14 and select "Save and Continue Editing" and 14 more blank spots will appear.
- Each row is a measurement, so if you have glucose and weight for some animal that is two rows entered.
- For animals, click on the magnifying glass then find the animal in that treatment group and click on the MouseID. The number displayed now in that field will not be the MouseID, but don't worry its just a different number to describe the mouse.
- For values, the standard units (defined by each assay) are mg for weights, mg/dL for glucose and pg/mL for insulin). You must enter integers here (no decimal places). If you have several measurements (ie several glucose readings during a GTT, enter them all in one measurement row, separated by commas and *NO spaces*).

CHAPTER

SIX

AUTOMATED DOCUMENTATION

6.1 Data Package

The data module describes the conditions and collection of data regarding experimental animals.

Data (or measurements) can be stored for any type of measurement. Conceptually, several pieces of data belong to an experiment (for example several mice are measured at some time) and several experiments belong to a study. Measurements can be stored independent of experiments and experiments can be performed outside of the context of a study. It is however, perfered that measurements are stored within an experiment and experiments are stored within studies as this will greatly facilitate the organization of the data.

6.1.1 Studies

In general studies are a collection of experiments. These can be grouped together on the basis of animals and/or treatment groups. A study must have at least one treatment group, which defines the animals and their conditions.

6.1.2 Experiments

An experiment is a collection of measurements for a given set of animals. In general, an experiment is defined as a number of measurements take in a given day.

6.1.3 Measurements

A measurement is an animal, an assay and a measurement value. It can be associated with an experiment, or can stand alone as an individual value. Measurements can be viewed in the context of a study, an experiment, a treatment group or an animal by going to the appropriate page.

6.1.4 Models

class data.models.**Assay**(**args*, ***kwargs*)

Bases: django.db.models.base.Model

Assay(id, assay, assay_slug, notes, measurement_units)

exception DoesNotExist

Bases: django.core.exceptions.ObjectDoesNotExist

exception Assay.**MultipleObjectsReturned**

Bases: django.core.exceptions.MultipleObjectsReturned

Assay.**measurement_set**

class data.models.**Diet**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Diet(id, vendor_id, description, product_id, fat_content, protein_content, carb_content, irradiated, notes)

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception Diet.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Diet.**treatment_set**

Diet.**vendor**

class data.models.**Environment**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Environment(id, building, room, temperature, humidity, notes)

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception Environment.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Environment.**contact**

Environment.**treatment_set**

class data.models.**Experiment**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Experiment(id, date, notes, experimentID, feeding_state, fasting_time, injection, concentration, study_id)

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception Experiment.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Experiment.**animals**

Experiment.**get_absolute_url**(**moreargs*, ***morekwargs*)

Experiment.**get_feeding_state_display**(**moreargs*, ***morekwargs*)

Experiment.**get_injection_display**(**moreargs*, ***morekwargs*)

Experiment.**get_next_by_date**(**moreargs*, ***morekwargs*)

Experiment.**get_previous_by_date**(**moreargs*, ***morekwargs*)

Experiment.**measurement_set**

Experiment.**researchers**

Experiment.**study**

class data.models.**Implantation**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Implantation(id, implant, vendor_id, product_id, notes)

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception Implantation.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Implantation.**surgeon**

Implantation.**treatment_set**

Implantation.**vendor**

class data.models.**Measurement**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Measurement(id, animal_id, experiment_id, assay_id, values)

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception Measurement.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Measurement.**age**()

Measurement.**animal**

Measurement.**assay**

Measurement.**experiment**

class data.models.**Pharmaceutical**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Pharmaceutical(id, drug, dose, recurrance, mode, vendor_id, notes)

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception Pharmaceutical.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Pharmaceutical.**get_mode_display**(**moreargs*, ***morekwargs*)

Pharmaceutical.**treatment_set**

Pharmaceutical.**vendor**

class data.models.**Researcher**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Researcher(id, first_name, last_name, name_slug, email, active)

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception Researcher.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Researcher.**environment_set**

Researcher.**experiment_set**

Researcher.**implantation_set**

Researcher.**transplantation_set**

Researcher.**treatment_set**

class data.models.**Study**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Study(id, description, start_date, stop_date, notes)

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception Study.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Study.**experiment_set**

Study.**get_absolute_url**(**moreargs*, ***morekwargs*)

Study.**strain**

Study.**treatment_set**

class data.models.**Transplantation**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Transplantation(id, tissue, transplant_date, notes)

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception Transplantation.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Transplantation.**donor**

Transplantation.**get_next_by_transplant_date**(**moreargs*, ***morekwargs*)

Transplantation.**get_previous_by_transplant_date**(**moreargs*, ***morekwargs*)

Transplantation.**surgeon**

Transplantation.**treatment_set**

class data.models.**Treatment**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Treatment(id, treatment, study_id, diet_id, environment_id, transplantation_id, notes)

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception Treatment.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Treatment.**animals**

Treatment.**diet**

Treatment.**environment**

Treatment.**get_absolute_url**(**moreargs*, ***morekwargs*)

Treatment.**implantation**

Treatment.**pharmaceutical**

Treatment.**researchers**

Treatment.**study**

Treatment.**transplantation**

class data.models.**Vendor**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Vendor(id, vendor, website, email, ordering, notes)

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception Vendor.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Vendor.**diet_set**

Vendor.**get_ordering_display**(**moreargs*, ***morekwargs*)

Vendor.**implantation_set**

Vendor.**pharmaceutical_set**

6.1.5 Forms

class data.forms.**ExperimentForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)
Bases: django.forms.models.ModelForm

This is the configuration for the experiment form.

This form is used to set up and modify an experiment. It uses a datepicker widget for the date, and autocomplete forms for the animals.

class Media

class ExperimentForm.**Meta**

model
alias of Experiment

ExperimentForm.**media**

class data.forms.**MeasurementForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)
Bases: django.forms.models.ModelForm

Form definition for adding and editing measurements.

This form is used for adding or modifying single measurements from within an experiment. It has an autocomplete field for animal.

class Media

class MeasurementForm.**Meta**

model
alias of Measurement

MeasurementForm.**media**

class data.forms.**StudyExperimentForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)

Bases: django.forms.models.ModelForm

This is the configuration for a study form (From an experiment).

This form provides an autocomplete field for the animals, and hides the study field which will be automatically set upon save.

class Media

class StudyExperimentForm.**Meta**

model
alias of Experiment

StudyExperimentForm.**media**

class data.forms.**StudyForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)

Bases: django.forms.models.ModelForm

This is the configuration for the study form.

This form is used to create and modify studies. It uses an autocomplete widget for the animals.

class Media

class StudyForm.**Meta**

model
alias of Study

StudyForm.**media**

class data.forms.**TreatmentForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)

Bases: django.forms.models.ModelForm

Form class for study treatment groups.

In the case of studies, animals are defined in the treatment group rather than in the study group. A treatment consists of a study, a set of animals and the conditions which define that treatment. This includes related fields for environment, diet, implants and transplants.

class Media

class TreatmentForm.**Meta**

model
alias of Treatment

TreatmentForm.**media**

6.1.6 Views and URLs

data.views.**add_measurement**(*request*, **args*, ***kwargs*)
This is a view to display a form to add single measurements to an experiment.

It calls the object MeasurementForm, which has an autocomplete field for animal.

data.views.**experiment_detail**(*request*, **args*, ***kwargs*)

data.views.**experiment_detail_all**(*request*, **args*, ***kwargs*)

data.views.**experiment_details_csv**(*request*, **args*, ***kwargs*)
This view generates a csv output file of an experiment.

The view writes to a csv table the animal, genotype, age (in days), assay and values.

data.views.**experiment_list**(*request*, **args*, ***kwargs*)

data.views.**study_experiment**(*request*, **args*, ***kwargs*)

6.1.7 Administrative Site Configuration

6.1.8 Test Files

This file contains tests for the data application.

These tests will verify generation of new experiment, measurement, assay, researcher, study, treatment, vendor, diet, environment, implantation, transplantation and pharnaceutical objects.

class data.tests.**StudyModelTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Test the creation and modification of Study objects.

setUp()
Instantiate the test client.

tearDown()
Depopulate created model instances from test database.

test_create_studey_detailed()
This is a test for creating a new study object, with all fields being entered. It also verifies that unicode is set correctly. This test is dependent on the ability to create a new Strain object (see animal.tests.StrainModelTests.test_create_minimal_strain).

test_create_study_minimal()
This is a test for creating a new study object, with only the minimum being entered. It also verifies that unicode is set correctly.

test_study_absolute_url()
This test verifies that the absolute url of a study object is set correctly. This study is dependend on a positive result on test_create_study_minimal.

class data.tests.**StudyViewTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

These tests test the views associated with Study objects.

setUp()
This function sets up the test client, and creates a test study.

tearDown()
Depopulate created model instances from test database.

test_study_delete()
This test checks the view which displays a study detail page. It checks for the correct templates and status code.

test_study_detail()
This test checks the view which displays a study detail page. It checks for the correct templates and status code.

test_study_edit()
This test checks the view which displays a study edit page. It checks for the correct templates and status code.

test_study_list()
This test checks the status code, and templates for study lists.

test_study_new()
This test checks the view which displays a study creation page. It checks for the correct templates and status code.

class data.tests.**TreatmentViewTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

These tests test the views associated with Treatment objects.

setUp()

tearDown()

test_treatment_detail()
This test checks the view which displays a treatment-detail page. It checks for the correct templates and status code.

6.2 Animals Package

The animal app contains and controls the display of data about animals.

Animals are tracked as individual entities, and given associations to breeding cages to follow ancestry, and strains.

6.2.1 Animal

Most parameters about an animal are set within the animal object. Here is where the animals strain, breeding, parentage and many other parameters are included. Animals have foreignkey relationships with both Strain and Breeding, so an animal may only belong to one of each of those. As an example, a mouse cannot come from more than one Breeding set, and cannot belong to more than one strain.

Backcrosses and Generations

For this software, optional tracking of backcrosses and generations is available and is stored as an attribute of an animal. When an inbred cross is made against a pure background, the backcross increases by 1. When a heterozygote cross is made, the generation increases by one. As an example, for every time a mouse in a C57/BL6 background is crossed against a wildtype C57/B6 mouse, the backcross (but not the generation) increases by one. For every time a mutant strain is crosses against itself (either vs a heterozygote or homozygote of that strain), the generation will

increase by one. Backcrosses should typically be performed against a separate colony of purebred mouse, rather than against wild-type alleles of the mutant strain.

6.2.2 Breeding Cages

A breeding cage is defined as a set of one or more male and one or more female mice. Because of this, it is not always clear who the precise parentage of an animal is. If the parentage is known, then the Mother and Father fields can be set for a particular animal.

6.2.3 Strains

A strain is a set of mice with a similar genetics. Importantly strains are separated from Backgrounds. For example, one might have mice with the genotype ob/ob but these mice may be in either a C57-Black6 or a mixed background. This difference is set at the individual animal level. The result of this is that a query for a particular strain may then need to be filtered to a specific background.

6.2.4 Models

This module describes the Strain, Animal, Breeding and Cage data models.

This module stores all data regarding a particular laboratory animal. Information about experimental data and timed matings are stored in the data and timed_matings packages. This module describes the database structure for each data model.

class `animal.models.`**`Strain`**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

A data model describing a mouse strain.

This is separate from the background of a mouse. For example a ob/ob mouse on a mixed or a black-6 background still have the same strain. The background is defined in the animal and breeding cages. Strain and Strain_slug are required.

class `animal.models.`**`Animal`**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

A data model describing an animal.

This data model describes a wide variety of parameters of an experimental animal. This model is linked to the Strain. If the parentage of a mouse is known, this can be identified (the breeding set may not be clear on this matter). Mice are automatically marked as not alive when a Death date is provided and the object is saved. Strain, Background and Genotype are required fields. By default, querysets are ordered first by strain then by MouseID.

class `animal.models.`**`Breeding`**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

This data model stores information about a particular breeding set

A breeding set may contain one ore more males and females and must be defined via the progeny strain. For example, in the case of generating a new strain, the strain indicates the new strain not the parental strains. A breeding cage is defined as one male with one or more females. If the breeding set is part of a timed mating experiment, then Timed_Mating must be selected. Breeding cages are automatically inactivated upon saving when a End date is provided. The only required field is Strain. By default, querysets are ordered by Strain, then Start.

6.2.5 Forms

Forms for use in manipulating objects in the animal app.

class animal.forms.**AnimalForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)

Bases: django.forms.models.ModelForm

This modelform provides fields for modifying animal data.

This form also automatically loads javascript and css for the datepicker jquery-ui widget. It also includes auto

class Media

class AnimalForm.**Meta**

model
alias of Animal

AnimalForm.**media**

class animal.forms.**BreedingForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)

Bases: django.forms.models.ModelForm

This form provides most fields for creating and entring breeding cage data.

This form is used from the url /mousedb/breeding/new and is a generic create view. This view includes a datepicker widget for Stat and End dates and autocomplete fields for the Females and Male fields

class Media

class BreedingForm.**Meta**

model
alias of Breeding

BreedingForm.**media**

class animal.forms.**MultipleAnimalForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)

Bases: django.forms.models.ModelForm

This modelform provides fields for entering multiple identical copies of a set of mice.

This form only includes the required fields Background and Strain.

class Meta

model
alias of Animal

MultipleAnimalForm.**media**

class animal.forms.**MultipleBreedingAnimalForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)

Bases: django.forms.models.ModelForm

This modelform provides fields for entering multiple pups within a breeding set.

The only fields presented are Born, Weaned, Gender and Count. Several other fields will be automatically entered based on the Breeding Set entries.

class Meta

model
alias of Animal

MultipleBreedingAnimalForm.**media**

6.2.6 Views and URLs

These views define template redirects for the animal app.

This module contains only non-generic views. Several generic views are also used and are defined in animal/urls/.

animal.views.**animal_detail**(*request*, **args*, ***kwargs*)
This view displays specific details about an animal.

It takes a request in the form animal/(id)/, mice/(id) or mouse/(id)/ and renders the detail page for that mouse. The request is defined for id not MouseID (or barcode) because this allows for details to be displayed for mice without barcode identification. Therefore care must be taken that animal/4932 is id=4932 and not barcode=4932. The animal name is defined at the top of the page. This page is restricted to logged-in users.

animal.views.**breeding_change**(*request*, **args*, ***kwargs*)
This view is used to generate a form by which to change pups which belong to a particular breeding set.

This view typically is used to modify existing pups. This might include marking animals as sacrificed, entering genotype or marking information or entering movement of mice to another cage. It is used to show and modify several animals at once. It takes a request in the form /breeding/(breeding_id)/change/ and returns a form specific to the breeding set defined in breeding_id. breeding_id is the background identification number of the breeding set and does not refer to the barcode of any breeding cage. This view returns a formset in which one row represents one animal. To add extra animals to a breeding set use /breeding/(breeding_id)/pups/. This view is restricted to those with the permission animal.change_animal.

animal.views.**breeding_detail**(*request*, **args*, ***kwargs*)
This view displays specific details about a breeding set.

It takes a request in the form /breeding/(breeding_id)/all and renders the detail page for that breeding set. The breeding_id refers to the background id of the breeding set, and not the breeding cage barcode. This view also passes along a dictionary of all pups belonging to that breeding set. This page is restricted to logged-in users.

animal.views.**breeding_pups**(*request*, **args*, ***kwargs*)
This view is used to generate a form by which to add pups which belong to a particular breeding set.

This view is intended to be used to add initial information about pups, including eartag, genotype, gender and birth or weaning information. It takes a request in the form /breeding/(breeding_id)/pups/ and returns a form specific to the breeding set defined in breeding_id. breeding_id is the background identification number of the

breeding set and does not refer to the barcode of any breeding cage. This view is restricted to those with the permission animal.add_animal.

animal.views.**breeding_wean**(*request*, **args*, ***kwargs*)
This view is used to generate a form by which to wean pups which belong to a particular breeding set.

This view typically is used to wean existing pups. This includes the MouseID, Cage, Markings, Gender and Wean Date fields. For other fields use the breeding-change page. It takes a request in the form /breeding/(breeding_id)/wean/ and returns a form specific to the breeding set defined in breeding_id. breeding_id is the background identification number of the breeding set and does not refer to the barcode of any breeding cage. This view returns a formset in which one row represents one animal. To add extra animals to a breeding set use /breeding/(breeding_id)/pups/. This view is restricted to those with the permission animal.change_animal.

animal.views.**date_archive_year**(*request*)
This view will generate a table of the number of mice born on an annual basis.

This view is associated with the url name archive-home, and returns an dictionary of a date and a animal count.

animal.views.**multiple_breeding_pups**(*request*, *breeding_id*)
This view is used to enter multiple animals at the same time from a breeding cage.

It will generate a form containing animal information and a number of mice. It is intended to create several identical anin
This view requres an input of a breeding_id to generate the correct form.

animal.views.**multiple_pups**(*request*)
This view is used to enter multiple animals at the same time.

It will generate a form containing animal information and a number of mice. It is intended to create several identical animals with the same attributes.

animal.views.**strain_detail**(*request*, **args*, ***kwargs*)
This view displays specific details about a strain.

It takes a request in the form strain/(strain_slug)/ and renders the detail page for that strain. This view also passes along a dictionary of alive animals belonging to that strain. This page is restricted to logged-in users.

animal.views.**strain_detail_all**(*request*, **args*, ***kwargs*)
This view displays specific details about a strain.

It takes a request in the form /strain/(strain_slug)/all and renders the detail page for that strain. This view also passes along a dictionary of all animals belonging to that strain. This page is restricted to logged-in users.

animal.views.**strain_list**(*request*, **args*, ***kwargs*)
This view presents a list of strains currently present in the database and annotates this list with a count of alive and total animals.

This view redirects from a /strain/ request. This view is restricted to logged-in users.

6.2.7 Administrative Site Configuration

Admin site settings for the animal app.

class animal.admin.**AnimalAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

Provides parameters for animal objects within the admin interface.

mark_sacrificed(*request*, *queryset*)
An admin action for marking several animals as sacrificed.

This action sets the selected animals as Alive=False, Death=today and Cause_of_Death as sacrificed. To use other paramters, mice muse be individually marked as sacrificed. This admin action also shows as the output the number of mice sacrificed.

media

class animal.admin.**AnimalInline**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

Provides an inline tabular formset for animal objects.

Currently used with the breeding admin page.

media

model
alias of Animal

class animal.admin.**BreedingAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

Settings in the admin interface for dealing with Breeding objects.

This interface also includes an form for adding objects associated with this breeding cage.

mark_deactivated(*request*, *queryset*)
An admin action for marking several cages as inactive.

This action sets the selected cages as Active=False and Death=today. This admin action also shows as the output the number of mice sacrificed.

media

class animal.admin.**StrainAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

Settings in the admin interface for dealing with Strain objects.

media

6.2.8 Test Files

This file contains tests for the animal application.

These tests will verify generation and function of a new breeding, strain and animal object.

class animal.tests.**AnimalModelTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Tests the model attributes of Animal objects contained in the animal app.

setUp()
Instantiate the test client.

tearDown()
Depopulate created model instances from test database.

test_animal_unicode()
This is a test for creating a new animal object, with only the minimum fields being entered. It then tests that the correct unicode representation is being generated.

test_create_animal_minimal()
This is a test for creating a new animal object, with only the minimum fields being entered

class animal.tests.**BreedingModelTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Tests the model attributes of Breeding objects contained in the animal app.

setUp()
Instantiate the test client.

tearDown()
Depopulate created model instances from test database.

test_autoset_active_state()
This is a test for creating a new breeding object, with only the minimum being entered. That object is then tested for the active state being automatically set when a End date is specified.

test_create_breeding_minimal()
This is a test for creating a new breeding object, with only the minimum being entered.

test_study_absolute_url()
This test verifies that the absolute url of a breeding object is set correctly.

test_unweaned()
This is a test for the unweaned animal list. It creates several animals for a breeding object and tests that they are tagged as unweaned. They are then weaned and retested to be tagged as not unweaned. This test is incomplete.

class animal.tests.**BreedingViewTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

These are tests for views based on Breeding objects. Included are tests for breeding list (active and all), details, create, update and delete pages as well as for the timed mating lists.

setUp()

tearDown()

test_breeding_delete()
This test checks the view which displays a breeding detail page. It checks for the correct templates and status code.

test_breeding_detail()
This test checks the view which displays a breeding detail page. It checks for the correct templates and status code.

test_breeding_edit()
This test checks the view which displays a breeding edit page. It checks for the correct templates and status code.

test_breeding_list()
This test checks the view which displays a breeding list page. It checks for the correct templates and status code.

test_breeding_list_all()
This test checks the view which displays a breeding list page, for all the cages. It checks for the correct templates and status code.

test_breeding_new()
This test checks the view which displays a new breeding page. It checks for the correct templates and status code.

test_timed_mating_list()
This test checks the view which displays a breeding list page, for all the cages. It checks for the correct templates and status code.

class animal.tests.**CageViewTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

These are tests for views based on animal objects as directed by cage urls. Included are tests for cage-list, cage-list-all and cage-detail

setUp()

tearDown()

test_cage_detail()
This test checks the view which displays a animal list page showing all animals with a specified cage number. It checks for the correct templates and status code.

test_cage_list()
This test checks the view which displays a cage list page showing all animals. It checks for the correct templates and status code.

class animal.tests.**DateViewTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

These are tests for views based on animal objects as directed by date based urls. Included are tests for archive-home, archive-month and archive-year

setUp()

tearDown()

test_archive_home()
This test checks the view which displays a summary of the birthdates of animals. It checks for the correct templates and status code.

test_archive_month()
This test checks the view which displays a list of the animals, filtered by month. It checks for the correct templates and status code.

test_archive_year()
This test checks the view which displays a list of the animals, filtered by year. It checks for the correct templates and status code.

6.3 Timed Mating Package

This package defines the timed_mating app.

Timed matings are a specific type of breeding set. Generally, for these experiments a mating cage is set up and pregnancy is defined by a plug event. Based on this information, the age of an embryo can be estimated. When a breeding cage is defined, one option is to set this cage as a timed mating cage (ie Timed_Mating=True). If this is the case, then a plug event can be registered and recorded for this mating set. If the mother gives birth then this cage is implicitly set as a normal breeding cage.

6.3.1 Models

This defines the data model for the timed_mating app.

Currently the only data model is for PlugEvents.

class timed_mating.models.**PlugEvents**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

This defines the model for PlugEvents.

A PlugEvent requires a date. All other fields are optional. Upon observation of a plug event, the PlugDate, Breeding Cage, Femalem, Male, Researcher and Notes can be set. Upon sacrifice of the mother, then genotyped alive and dead embryos can be entered, along with the SacrificeDate, Researcher and Notes.

Breeding

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception PlugEvents.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

PlugEvents.**PlugFemale**

PlugEvents.**PlugMale**

PlugEvents.**Researcher**

PlugEvents.**get_absolute_url**(**moreargs*, ***morekwargs*)

PlugEvents.**get_next_by_PlugDate**(**moreargs*, ***morekwargs*)

PlugEvents.**get_previous_by_PlugDate**(**moreargs*, ***morekwargs*)

PlugEvents.**save**()
Over-rides the default save function for PlugEvents.

If a sacrifice date is set for an object in this model, then Active is set to False.

class timed_mating.models.**PlugEvents**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

This defines the model for PlugEvents.

A PlugEvent requires a date. All other fields are optional. Upon observation of a plug event, the PlugDate, Breeding Cage, Femalem, Male, Researcher and Notes can be set. Upon sacrifice of the mother, then genotyped alive and dead embryos can be entered, along with the SacrificeDate, Researcher and Notes.

6.3.2 Forms

This package describes forms used by the Timed Mating app.

class timed_mating.forms.**BreedingPlugForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)
Bases: django.forms.models.ModelForm

This form is used to enter Plug Events from a specific breeding cage.

class Meta

model
alias of PlugEvents

BreedingPlugForm.**media**

6.3.3 Views and URLs

This package defines custom views for the timed_mating application.

Currently all views are generic CRUD views except for the view in which a plug event is defined from a breeding cage.

timed_mating.views.**breeding_plugevent**(*request*, **args*, ***kwargs*)
This view defines a form for adding new plug events from a breeding cage.

This form requires a breeding_id from a breeding set and restricts the PlugFemale and PlugMale to animals that are defined in that breeding cage.

This urlconf sets the directions for the timed_mating app.

It comprises of create, update, delete, detail and list of plug events.

timed_mating.urls.**change_plugevents**(*request*, **args*, ***kwargs*)

timed_mating.urls.**create_plugevents**(*request*, **args*, ***kwargs*)

timed_mating.urls.**delete_plugevents**(*request*, **args*, ***kwargs*)

timed_mating.urls.**limited_object_detail**(*request*, **args*, ***kwargs*)

timed_mating.urls.**limited_object_list**(*request*, **args*, ***kwargs*)

6.3.4 Administrative Site Configuration

Settings to control the admin interface for the timed_mating app.

This file defines a PlugEventsAdmin object to enter parameters about individual plug events/

class timed_mating.admin.**PlugEventsAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

This class defines the admin interface for the PlugEvents model.

media

6.3.5 Test Files

This file contains tests for the timed_mating application.

These tests will verify generation of a new PlugEvent object.

class timed_mating.tests.**Timed_MatingModelTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Test the models contained in the 'timed_mating' app.

setUp()
Instantiate the test client. Creates a test user.

tearDown()
Depopulate created model instances from test database.

test_create_plugevent_minimal()
This is a test for creating a new PlugEvent object, with only the minimum being entered.

test_create_plugevent_most_fields()
This is a test for creating a new PlugEvent object.

This test uses a Breeding, PlugDate, PlugMale and PlugFemale field.

test_set_plugevent_inactive()
This is a test for the automatic inactivation of a cage when the SacrificeDate is entered.

class timed_mating.tests.**Timed_MatingViewTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Test the views contained in the 'timed_mating' app.

setUp()
Instantiate the test client. Creates a test user.

tearDown()
Depopulate created model instances from test database.

test_plugevent_delete()
This tests the plugevent-delete view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_plugevent_detail()
This tests the plugevent-detail view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_plugevent_edit()
This tests the plugevent-edit view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_plugevent_list()
This tests the plugevent-list view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_plugevent_new()
This tests the plugevent-new view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

test_plugeventbreeding_new()
This tests the plugevent-new view, ensuring that templates are loaded correctly.

This view uses a user with superuser permissions so does not test the permission levels for this view.

6.4 Groups Package

This package defines the Group application. This app defines generic Group and License information for a particular installation of MouseDB. Because every page on this site identifies both the Group and data restrictions, at a minimum, group information must be provided upon installation (see installation instructions).

6.4.1 Models

class groups.models.**Group**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

This defines the data structure for the Group model.

The only required field is group. All other fields (group_slug, group_url, license, contact_title, contact_first, contact_last and contact_email) are optional.

exception `DoesNotExist`
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception `Group.MultipleObjectsReturned`
Bases: `django.core.exceptions.MultipleObjectsReturned`

`Group.get_contact_title_display(*moreargs, **morekwargs)`

`Group.license`

class `groups.models.License(*args, **kwargs)`
Bases: `django.db.models.base.Model`

This defines the data structure for the License model.

The only required field is license. If the contents of this installation are being made available using some licencing criteria this can either be defined in the notes field, or in an external website.

exception `DoesNotExist`
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception `License.MultipleObjectsReturned`
Bases: `django.core.exceptions.MultipleObjectsReturned`

`License.group_set`

class `groups.models.Group(*args, **kwargs)`
Bases: `django.db.models.base.Model`

This defines the data structure for the Group model.

The only required field is group. All other fields (group_slug, group_url, license, contact_title, contact_first, contact_last and contact_email) are optional.

6.4.2 Views and URLs

6.4.3 Administrative Site Configuration

class `groups.admin.GroupAdmin(model, admin_site)`
Bases: `django.contrib.admin.options.ModelAdmin`

Defines the admin interface for Groups.

Currently set as default.

`media`

class `groups.admin.LicenseAdmin(model, admin_site)`
Bases: `django.contrib.admin.options.ModelAdmin`

Defines the admin interface for Licences.

Currently set as default.

`media`

6.4.4 Test Files

This file contains tests for the groups application.

These tests will verify generation of a new group and license object.

class groups.tests.**GroupsModelTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Test the models contained in the 'groups' app.

setUp()
Instantiate the test client.

tearDown()
Depopulate created model instances from test database.

test_create_group_all_fields()
This is a test for creating a new group object, with all fields being entered, except license.

test_create_group_minimal()
This is a test for creating a new group object, with only the minimum being entered.

test_create_license_all_fields()
This is a test for creating a new license object, with all fields being entered.

test_create_license_minimal()
This is a test for creating a new license object, with only the minimum being entered.

CHAPTER

SEVEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

a

animal, 22
animal.admin, 26
animal.forms, 24
animal.models, 23
animal.tests, 27
animal.urls, 26
animal.views, 25

d

data, 15
data.admin, 21
data.forms, 19
data.models, 15
data.tests, 21
data.urls, 21
data.views, 21

g

groups, 32
groups.admin, 33
groups.models, 32
groups.tests, 33
groups.views, 33

t

timed_mating, 29
timed_mating.admin, 31
timed_mating.forms, 30
timed_mating.models, 29
timed_mating.tests, 31
timed_mating.urls, 31
timed_mating.views, 31

INDEX

A

add_measurement() (in module data.views), 21
age() (data.models.Measurement method), 17
Animal (class in animal.models), 23
animal (data.models.Measurement attribute), 17
animal (module), 22
animal.admin (module), 26
animal.forms (module), 24
animal.models (module), 23
animal.tests (module), 27
animal.urls (module), 26
animal.views (module), 25
animal_detail() (in module animal.views), 25
AnimalAdmin (class in animal.admin), 26
AnimalForm (class in animal.forms), 24
AnimalForm.Media (class in animal.forms), 24
AnimalForm.Meta (class in animal.forms), 24
AnimalInline (class in animal.admin), 27
AnimalModelTests (class in animal.tests), 27
animals (data.models.Experiment attribute), 16
animals (data.models.Treatment attribute), 18
Assay (class in data.models), 15
assay (data.models.Measurement attribute), 17
Assay.DoesNotExist, 15
Assay.MultipleObjectsReturned, 15

B

Breeding (class in animal.models), 23
Breeding (timed_mating.models.PlugEvents attribute), 30
breeding_change() (in module animal.views), 25
breeding_detail() (in module animal.views), 25
breeding_plugevent() (in module timed_mating.views), 31
breeding_pups() (in module animal.views), 25
breeding_wean() (in module animal.views), 26
BreedingAdmin (class in animal.admin), 27
BreedingForm (class in animal.forms), 24
BreedingForm.Media (class in animal.forms), 24
BreedingForm.Meta (class in animal.forms), 24
BreedingModelTests (class in animal.tests), 27
BreedingPlugForm (class in timed_mating.forms), 30
BreedingPlugForm.Meta (class in timed_mating.forms), 30
BreedingViewTests (class in animal.tests), 28

C

CageViewTests (class in animal.tests), 28
change_plugevents() (in module timed_mating.urls), 31
contact (data.models.Environment attribute), 16
create_plugevents() (in module timed_mating.urls), 31

D

data (module), 15
data.admin (module), 21
data.forms (module), 19
data.models (module), 15
data.tests (module), 21
data.urls (module), 21
data.views (module), 21
date_archive_year() (in module animal.views), 26
DateViewTests (class in animal.tests), 29
delete_plugevents() (in module timed_mating.urls), 31
Diet (class in data.models), 16
diet (data.models.Treatment attribute), 18
Diet.DoesNotExist, 16
Diet.MultipleObjectsReturned, 16
diet_set (data.models.Vendor attribute), 19
donor (data.models.Transplantation attribute), 18

E

Environment (class in data.models), 16
environment (data.models.Treatment attribute), 18
Environment.DoesNotExist, 16
Environment.MultipleObjectsReturned, 16
environment_set (data.models.Researcher attribute), 17
Experiment (class in data.models), 16
experiment (data.models.Measurement attribute), 17
Experiment.DoesNotExist, 16
Experiment.MultipleObjectsReturned, 16
experiment_detail() (in module data.views), 21
experiment_detail_all() (in module data.views), 21
experiment_details_csv() (in module data.views), 21

experiment_list() (in module data.views), 21
experiment_set (data.models.Researcher attribute), 17
experiment_set (data.models.Study attribute), 18
ExperimentForm (class in data.forms), 19
ExperimentForm.Media (class in data.forms), 19
ExperimentForm.Meta (class in data.forms), 19

G

get_absolute_url() (data.models.Experiment method), 16
get_absolute_url() (data.models.Study method), 18
get_absolute_url() (data.models.Treatment method), 18
get_absolute_url() (timed_mating.models.PlugEvents method), 30
get_contact_title_display() (groups.models.Group method), 33
get_feeding_state_display() (data.models.Experiment method), 16
get_injection_display() (data.models.Experiment method), 16
get_mode_display() (data.models.Pharmaceutical method), 17
get_next_by_date() (data.models.Experiment method), 16
get_next_by_PlugDate() (timed_mating.models.PlugEvents method), 30
get_next_by_transplant_date() (data.models.Transplantation method), 18
get_ordering_display() (data.models.Vendor method), 19
get_previous_by_date() (data.models.Experiment method), 16
get_previous_by_PlugDate() (timed_mating.models.PlugEvents method), 30
get_previous_by_transplant_date() (data.models.Transplantation method), 18
Group (class in groups.models), 32, 33
Group.DoesNotExist, 32
Group.MultipleObjectsReturned, 33
group_set (groups.models.License attribute), 33
GroupAdmin (class in groups.admin), 33
groups (module), 32
groups.admin (module), 33
groups.models (module), 32
groups.tests (module), 33
groups.views (module), 33
GroupsModelTests (class in groups.tests), 33

I

Implantation (class in data.models), 16
implantation (data.models.Treatment attribute), 18
Implantation.DoesNotExist, 16
Implantation.MultipleObjectsReturned, 16
implantation_set (data.models.Researcher attribute), 17
implantation_set (data.models.Vendor attribute), 19

L

License (class in groups.models), 33
license (groups.models.Group attribute), 33
License.DoesNotExist, 33
License.MultipleObjectsReturned, 33
LicenseAdmin (class in groups.admin), 33
limited_object_detail() (in module timed_mating.urls), 31
limited_object_list() (in module timed_mating.urls), 31

M

mark_deactivated() (animal.admin.BreedingAdmin method), 27
mark_sacrificed() (animal.admin.AnimalAdmin method), 26
Measurement (class in data.models), 17
Measurement.DoesNotExist, 17
Measurement.MultipleObjectsReturned, 17
measurement_set (data.models.Assay attribute), 16
measurement_set (data.models.Experiment attribute), 16
MeasurementForm (class in data.forms), 19
MeasurementForm.Media (class in data.forms), 19
MeasurementForm.Meta (class in data.forms), 19
media (animal.admin.AnimalAdmin attribute), 27
media (animal.admin.AnimalInline attribute), 27
media (animal.admin.BreedingAdmin attribute), 27
media (animal.admin.StrainAdmin attribute), 27
media (animal.forms.AnimalForm attribute), 24
media (animal.forms.BreedingForm attribute), 24
media (animal.forms.MultipleAnimalForm attribute), 24
media (animal.forms.MultipleBreedingAnimalForm attribute), 25
media (data.forms.ExperimentForm attribute), 19
media (data.forms.MeasurementForm attribute), 19
media (data.forms.StudyExperimentForm attribute), 20
media (data.forms.StudyForm attribute), 20
media (data.forms.TreatmentForm attribute), 20
media (groups.admin.GroupAdmin attribute), 33
media (groups.admin.LicenseAdmin attribute), 33
media (timed_mating.admin.PlugEventsAdmin attribute), 31
media (timed_mating.forms.BreedingPlugForm attribute), 30
model (animal.admin.AnimalInline attribute), 27
model (animal.forms.AnimalForm.Meta attribute), 24
model (animal.forms.BreedingForm.Meta attribute), 24
model (animal.forms.MultipleAnimalForm.Meta attribute), 24
model (animal.forms.MultipleBreedingAnimalForm.Meta attribute), 25
model (data.forms.ExperimentForm.Meta attribute), 19
model (data.forms.MeasurementForm.Meta attribute), 19
model (data.forms.StudyExperimentForm.Meta attribute), 20
model (data.forms.StudyForm.Meta attribute), 20

model (data.forms.TreatmentForm.Meta attribute), 20
model (timed_mating.forms.BreedingPlugForm.Meta attribute), 30
multiple_breeding_pups() (in module animal.views), 26
multiple_pups() (in module animal.views), 26
MultipleAnimalForm (class in animal.forms), 24
MultipleAnimalForm.Meta (class in animal.forms), 24
MultipleBreedingAnimalForm (class in animal.forms), 24
MultipleBreedingAnimalForm.Meta (class in animal.forms), 25

P

Pharmaceutical (class in data.models), 17
pharmaceutical (data.models.Treatment attribute), 18
Pharmaceutical.DoesNotExist, 17
Pharmaceutical.MultipleObjectsReturned, 17
pharmaceutical_set (data.models.Vendor attribute), 19
PlugEvents (class in timed_mating.models), 29, 30
PlugEvents.DoesNotExist, 30
PlugEvents.MultipleObjectsReturned, 30
PlugEventsAdmin (class in timed_mating.admin), 31
PlugFemale (timed_mating.models.PlugEvents attribute), 30
PlugMale (timed_mating.models.PlugEvents attribute), 30

R

Researcher (class in data.models), 17
Researcher (timed_mating.models.PlugEvents attribute), 30
Researcher.DoesNotExist, 17
Researcher.MultipleObjectsReturned, 17
researchers (data.models.Experiment attribute), 16
researchers (data.models.Treatment attribute), 18

S

save() (timed_mating.models.PlugEvents method), 30
setUp() (animal.tests.AnimalModelTests method), 27
setUp() (animal.tests.BreedingModelTests method), 28
setUp() (animal.tests.BreedingViewTests method), 28
setUp() (animal.tests.CageViewTests method), 29
setUp() (animal.tests.DateViewTests method), 29
setUp() (data.tests.StudyModelTests method), 21
setUp() (data.tests.StudyViewTests method), 21
setUp() (data.tests.TreatmentViewTests method), 22
setUp() (groups.tests.GroupsModelTests method), 34
setUp() (timed_mating.tests.Timed_MatingModelTests method), 31
setUp() (timed_mating.tests.Timed_MatingViewTests method), 32
Strain (class in animal.models), 23
strain (data.models.Study attribute), 18
strain_detail() (in module animal.views), 26
strain_detail_all() (in module animal.views), 26
strain_list() (in module animal.views), 26
StrainAdmin (class in animal.admin), 27
Study (class in data.models), 17
study (data.models.Experiment attribute), 16
study (data.models.Treatment attribute), 18
Study.DoesNotExist, 18
Study.MultipleObjectsReturned, 18
study_experiment() (in module data.views), 21
StudyExperimentForm (class in data.forms), 19
StudyExperimentForm.Media (class in data.forms), 20
StudyExperimentForm.Meta (class in data.forms), 20
StudyForm (class in data.forms), 20
StudyForm.Media (class in data.forms), 20
StudyForm.Meta (class in data.forms), 20
StudyModelTests (class in data.tests), 21
StudyViewTests (class in data.tests), 21
surgeon (data.models.Implantation attribute), 17
surgeon (data.models.Transplantation attribute), 18

T

tearDown() (animal.tests.AnimalModelTests method), 27
tearDown() (animal.tests.BreedingModelTests method), 28
tearDown() (animal.tests.BreedingViewTests method), 28
tearDown() (animal.tests.CageViewTests method), 29
tearDown() (animal.tests.DateViewTests method), 29
tearDown() (data.tests.StudyModelTests method), 21
tearDown() (data.tests.StudyViewTests method), 21
tearDown() (data.tests.TreatmentViewTests method), 22
tearDown() (groups.tests.GroupsModelTests method), 34
tearDown() (timed_mating.tests.Timed_MatingModelTests method), 31
tearDown() (timed_mating.tests.Timed_MatingViewTests method), 32
test_animal_unicode() (animal.tests.AnimalModelTests method), 27
test_archive_home() (animal.tests.DateViewTests method), 29
test_archive_month() (animal.tests.DateViewTests method), 29
test_archive_year() (animal.tests.DateViewTests method), 29
test_autoset_active_state() (animal.tests.BreedingModelTests method), 28
test_breeding_delete() (animal.tests.BreedingViewTests method), 28
test_breeding_detail() (animal.tests.BreedingViewTests method), 28
test_breeding_edit() (animal.tests.BreedingViewTests method), 28
test_breeding_list() (animal.tests.BreedingViewTests method), 28

test_breeding_list_all() (animal.tests.BreedingViewTests method), 28
test_breeding_new() (animal.tests.BreedingViewTests method), 28
test_cage_detail() (animal.tests.CageViewTests method), 29
test_cage_list() (animal.tests.CageViewTests method), 29
test_create_animal_minimal() (animal.tests.AnimalModelTests method), 27
test_create_breeding_minimal() (animal.tests.BreedingModelTests method), 28
test_create_group_all_fields() (groups.tests.GroupsModelTests method), 34
test_create_group_minimal() (groups.tests.GroupsModelTests method), 34
test_create_license_all_fields() (groups.tests.GroupsModelTests method), 34
test_create_license_minimal() (groups.tests.GroupsModelTests method), 34
test_create_plugevent_minimal() (timed_mating.tests.Timed_MatingModelTests method), 31
test_create_plugevent_most_fields() (timed_mating.tests.Timed_MatingModelTests method), 31
test_create_studey_detailed() (data.tests.StudyModelTests method), 21
test_create_study_minimal() (data.tests.StudyModelTests method), 21
test_plugevent_delete() (timed_mating.tests.Timed_MatingViewTests method), 32
test_plugevent_detail() (timed_mating.tests.Timed_MatingViewTests method), 32
test_plugevent_edit() (timed_mating.tests.Timed_MatingViewTests method), 32
test_plugevent_list() (timed_mating.tests.Timed_MatingViewTests method), 32
test_plugevent_new() (timed_mating.tests.Timed_MatingViewTests method), 32
test_plugeventbreeding_new() (timed_mating.tests.Timed_MatingViewTests method), 32
test_set_plugevent_inactive() (timed_mating.tests.Timed_MatingModelTests method), 31
test_study_absolute_url() (animal.tests.BreedingModelTests method), 28
test_study_absolute_url() (data.tests.StudyModelTests method), 21
test_study_delete() (data.tests.StudyViewTests method), 22
test_study_detail() (data.tests.StudyViewTests method), 22
test_study_edit() (data.tests.StudyViewTests method), 22
test_study_list() (data.tests.StudyViewTests method), 22
test_study_new() (data.tests.StudyViewTests method), 22
test_timed_mating_list() (animal.tests.BreedingViewTests method), 28
test_treatment_detail() (data.tests.TreatmentViewTests method), 22
test_unweaned() (animal.tests.BreedingModelTests method), 28
timed_mating (module), 29
timed_mating.admin (module), 31
timed_mating.forms (module), 30
timed_mating.models (module), 29
timed_mating.tests (module), 31
timed_mating.urls (module), 31
timed_mating.views (module), 31
Timed_MatingModelTests (class in timed_mating.tests), 31
Timed_MatingViewTests (class in timed_mating.tests), 32
Transplantation (class in data.models), 18
transplantation (data.models.Treatment attribute), 18
Transplantation.DoesNotExist, 18
Transplantation.MultipleObjectsReturned, 18
transplantation_set (data.models.Researcher attribute), 17
Treatment (class in data.models), 18
Treatment.DoesNotExist, 18
Treatment.MultipleObjectsReturned, 18
treatment_set (data.models.Diet attribute), 16
treatment_set (data.models.Environment attribute), 16
treatment_set (data.models.Implantation attribute), 17
treatment_set (data.models.Pharmaceutical attribute), 17
treatment_set (data.models.Researcher attribute), 17
treatment_set (data.models.Study attribute), 18
treatment_set (data.models.Transplantation attribute), 18
TreatmentForm (class in data.forms), 20
TreatmentForm.Media (class in data.forms), 20
TreatmentForm.Meta (class in data.forms), 20
TreatmentViewTests (class in data.tests), 22

V

Vendor (class in data.models), 18
vendor (data.models.Diet attribute), 16
vendor (data.models.Implantation attribute), 17
vendor (data.models.Pharmaceutical attribute), 17
Vendor.DoesNotExist, 19
Vendor.MultipleObjectsReturned, 19