pylabber

Release 0.1.0

Jun 08, 2022

Contents:

pylabber is a Django project designed to provide the core functionality required to conduct neuroscientific research.

See also:

For more information about Django applications and the difference between projects and apps see Django's Getting Started page, as well as the Projects and Applications section.

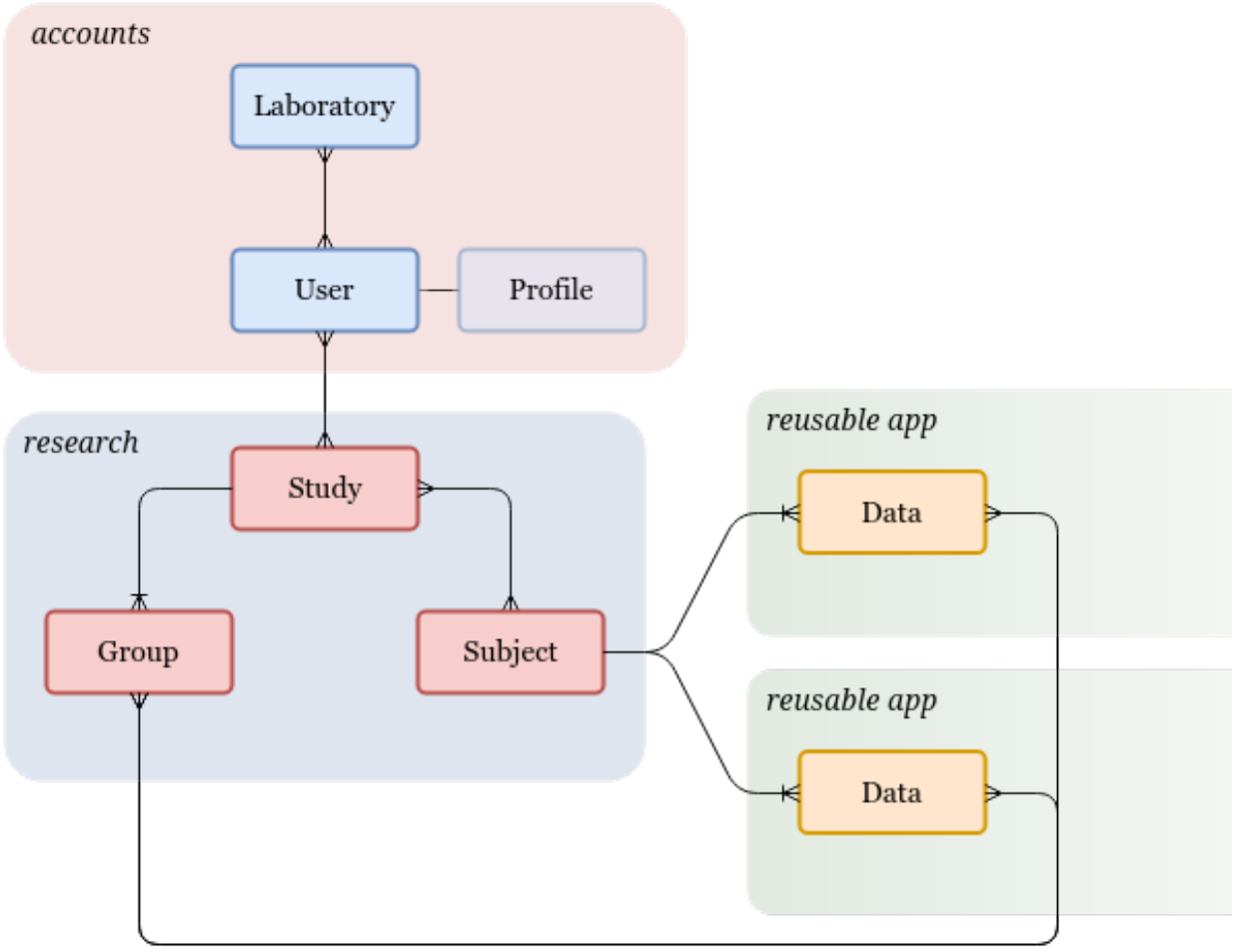
CHAPTER 1

Overview

The project's root directory contains the *pylabber* module, which holds the applications settings and URL configurations, as well as two native apps:

- *accounts*: Manages *User* (i.e. researcher) and *Laboratory* accounts.
- *research*: Manages the most elementary "research entities"; *Subject*, *Study*, and *Group*.

The basic models detailed above are meant to be integrated with external, reusable apps, providing domain-specific data models and functionality over them.



For a list of integrated reusable apps see *Existing Apps*.

CHAPTER 2

Installation

Note: *pylabber* requires PostgreSQL. Before starting the installation, make sure you have PostgreSQL installed and create a new database.

There are many tutorials online providing detailed instructions on how to do this, some recommended ones can be found in Digital Ocean, DjangoGirls, and Medium.

1. Clone *pylabber* to a local directory:

```
$ git clone https://github.com/TheLabbingProject/pylabber
```

2. From within the project's directory, create a virtual environment using venv:

```
$ python3 -m venv venv
```

3. Activate the virtual environment:

```
$ source venv/bin/activate
```

4. Install the required dependencies:

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

5. Create the appropriate environment variables for the *settings.py* file. You can find each of those variables wrapped in an *env("VAR_NAME")* call.

An example for an environment variables definition in development could look like:

Listing 1: .env

```
export SECRET_KEY="s0m3-$\/pER-s3CrE7_kEy!"
export DB_NAME=pylabber
export DB_USER=postgres
```

(continues on next page)

(continued from previous page)

```
export DB_PASSWORD=p$q1@ddmmIN
export MEDIA_ROOT="/where/to/save/files/locally/"
```

6. Apply the project's migrations:

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

7. Run *pylabber* locally:

```
$ python manage.py runserver
```

or, open an interactive Django shell using django-extensions:

```
$ python manage.py shell_plus
```

CHAPTER 3

Existing Apps

- django_analyses: Database-driven pipeline engine. Allows the creation of a library of analyses and keeps records of configurations and runs.
- django_mri: Format-agnostic MRI data management. Provides a `Scan` model that abstractifies data management and retrieval.

CHAPTER 4

Frequently Asked Questions

4.1 What is The Labbing Project?

A collection of reusable Django apps and a single dedicated parent Django project (i.e. pylabber) used to manage and share research data and derivatives across researchers and across labs.

Hint: For more information about the difference between Django apps and projects, see Django's Projects and Applications documentation.

4.2 Is The Labbing Project only suitable for MRI-based neuroscientific research?

Not at all! The purpose of the `research` app is to provide common, generic models (such as `Subject` and `Study`) that may be associted with any number of data models originating from any number of reusable apps.

4.3 How are different data types integrated into the application's database?

The modular design of the project enables researchers from different fields to create specialized apps that manage the relevant data formats and facilitate commonplace workflows. E.g. the first app created for this purpose was django_dicom, which manages DICOM data, and the second was django_mri, providing a format-agnostic `Scan` abstraction, as well as other useful models and MRI-based research utilities. Each reusable app integrates with pylabber's internal *`research`* app to associate the data with an instance of the *`Subject`* model. For more information, see pylabber's overview.

CHAPTER 5

Reference

5.1 accounts package

5.1.1 Module contents

Manages researcher (`User`) and `Laboratory` accounts.

5.1.2 Subpackages

accounts.filters package

Module contents

Filters for *the app's models*.

Notes

For more information, see:

- Django REST Framework filtering documentation.
- django-filter's documentation for Integration with DRF.

Submodules

accounts.filters.user module

Definition of the `UserFilter` class.

class accounts.filters.user.**UserFilter**(*data=None*, *queryset=None*, ***, *request=None*, *prefix=None*)

Bases: django_filters.rest_framework.filterset.FilterSet

Provides useful filtering options for the *User* model.

base_filters = {'email': <django_filters.filters.LookupChoiceFilter object>, 'first_n

declared_filters = {'email': <django_filters.filters.LookupChoiceFilter object>, 'firs

accounts.models package

Module contents

Definition of the app's models. For an illustration of the relationship between the different models, see the *Overview*.

Submodules

accounts.models.choices module

Subclasses of the ChoiceEnum class used to represent raw and human-readable values for choices within the *accounts.models* module.

class accounts.models.choices.**Role**

Bases: pylabber.utils.utils.ChoiceEnum

An enumeration.

AFF = 'Research Affiliate'

MAN = 'Lab Manager'

MSC = 'M.Sc. Student'

PHD = 'Ph.D. Candidate'

PI = 'Principle Investigator'

POST = 'Postdoctoral Researcher'

RA = 'Research Assistant'

class accounts.models.choices.**Title**

Bases: pylabber.utils.utils.ChoiceEnum

An enumeration.

BSC = 'B.Sc.'

MSC = 'M.Sc.'

PHD = 'Ph.D.'

PROF = 'Prof.'

accounts.models.laboratory module

Definition of the *Laboratory* model.

class accounts.models.laboratory.**Laboratory**(**args*, ***kwargs*)
Bases: django_extensions.db.models.TitleDescriptionModel, django_extensions.db.models.TimeStampedModel

A class to represents a research laboratory.

get_absolute_url() → str
Returns the canonical URL for this instance.

References

- get_absolute_url()

Returns URL

Return type str

image
Just like the FileDescriptor, but for ImageFields. The only difference is assigning the width/height to the width_field/height_field, if appropriate.

laboratorymembership_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

members
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

accounts.models.profile module

Definition of the *Profile* model.

class accounts.models.profile.**Profile**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

A user profile, associated to each user using a OneToOne relationship and created automatically usings signals.

References

- Extending the User model.

bio
Short user biography.

date_of_birth
User's date of birth.

get_absolute_url() → str
Returns the canonical URL for this instance.

References

- get_absolute_url()

Returns URL

Return type str

get_full_name(*include_title: bool = True*) → str
Returns the full name of the user, including a title if any.

Parameters **include_title** (*bool, optional*) – Whether to include title (academic credentials etc.) if specified

Returns User's full name

Return type str

get_title_display(*, *field=<django.db.models.fields.CharField: title>*)

get_title_repr() → str
Returns the verbose title of the user as defined in the *Title* Enum.

Returns Verbose title representation

Return type str

image
Profile image.

institute
The institute to which a user belongs, if any.

title
Academic or any other kind of title.

user
OneToOne relationship with the user model.

accounts.models.user module

Definition of the *User* model.

class accounts.models.user.**User**(**args*, ***kwargs*)
Bases: django.contrib.auth.models.AbstractUser

Custom User model definition.

References

- Using a custom user model when starting a project.

exportdestination_set
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

groups
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

laboratory_set
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

laboratorymembership_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

logentry_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

mri_uploads
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

profile
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

run_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

study_set
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

user_permissions
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

accounts.serializers package

Module contents

Django REST Framework serializers module for the *accounts* package.

Submodules

accounts.serializers.group module

Definition of the *GroupSerializer* class.

class accounts.serializers.group.**GroupSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: `rest_framework.serializers.HyperlinkedModelSerializer`

Serializer class for the `Group` model.

accounts.serializers.laboratory module

Definition of the *LaboratorySerializer* class.

class accounts.serializers.laboratory.**LaboratorySerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: `rest_framework.serializers.HyperlinkedModelSerializer`

Serializer class for the *Laboratory* model.

accounts.serializers.profile module

Definition of the *ProfileSerializer* class.

class accounts.serializers.profile.**ProfileSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: `rest_framework.serializers.HyperlinkedModelSerializer`

Serializer class for the *Profile* model.

accounts.serializers.user module

Definition of the *UserSerializer* class.

class accounts.serializers.user.**UserSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: rest_auth.serializers.UserDetailsSerializer

Serializer class for the *User* model.

References

- https://www.django-rest-framework.org/api-guide/serializers/

update(*username*, *data: dict*)

Update a user's personal information, including profile data.

Parameters

- **username** (*User*) – User to be updated
- **data** (*dict*) – User information

Returns Updated user instance

Return type *User*

accounts.views package

Module contents

Django REST Framework ViewSets for the *accounts* package.

Submodules

accounts.views.group module

Definition of the *GroupViewSet* class.

class accounts.views.group.**GroupViewSet**(***kwargs*)

Bases: pylabber.views.defaults.DefaultsMixin, rest_framework.viewsets.ModelViewSet

API endpoint that allows Group instances to be viewed or edited.

basename = None

description = None

detail = None

name = None

queryset

serializer_class

alias of *accounts.serializers.group.GroupSerializer*

suffix = None

accounts.views.laboratory module

Definition of the *LaboratoryViewSet* class.

class accounts.views.laboratory.**LaboratoryViewSet**(***kwargs*)
Bases: pylabber.views.defaults.DefaultsMixin, rest_framework.viewsets.ModelViewSet

API endpoint that allows *Laboratory* instances to be viewed or edited.

basename = None

description = None

detail = None

name = None

queryset

serializer_class
alias of *accounts.serializers.laboratory.LaboratorySerializer*

suffix = None

accounts.views.profile module

Definition of the *ProfileViewSet* class.

class accounts.views.profile.**ProfileViewSet**(***kwargs*)
Bases: pylabber.views.defaults.DefaultsMixin, rest_framework.viewsets.ModelViewSet

API endpoint that allows *Profile* instances to be viewed or edited.

basename = None

description = None

detail = None

name = None

queryset

serializer_class
alias of *accounts.serializers.profile.ProfileSerializer*

suffix = None

accounts.views.user module

Definition of the *UserViewSet* class.

class accounts.views.user.**UserViewSet**(***kwargs*)
Bases: pylabber.views.defaults.DefaultsMixin, rest_framework.viewsets.ModelViewSet

API endpoint that allows *User* instances to be viewed or edited.

basename = None

description = None

detail = None

filter_class
alias of *accounts.filters.user.UserFilter*

filter_queryset (*queryset*) → django.db.models.query.QuerySet
Filter the returned users according to the requesting user's permissions.

Parameters **queryset** (*QuerySet*) – Base queryset

Returns User instances

Return type QuerySet

get_institutions (*request*)

name = None

queryset

serializer_class
alias of *accounts.serializers.user.UserSerializer*

suffix = None

5.1.3 Submodules

5.1.4 accounts.admin module

class accounts.admin.**ExportDestinationAdmin** (*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

fieldsets = ((None, {'fields': ('id', 'title', 'description', 'username', 'password',

form
alias of accounts.forms.export_destination.ExportDestinationForm

list_display = ('id', 'title', 'description', 'ip', 'username', 'destination', 'sftp')

media

readonly_fields = ('id',)

sftp (*destination: accounts.models.export_destination.ExportDestination*) → bool

class accounts.admin.**LaboratoryAdmin** (*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

class Media
Bases: object

css = {'all': ('accounts/css/hide_admin_original.css',)}

fields = ('title', 'description', 'image', 'image_tag')

image_tag (*instance: accounts.models.laboratory.Laboratory*) → str

inlines = (<class 'accounts.admin.LaboratoryMembershipInLine'>,)

list_display = ('id', 'title', 'description', 'created', 'modified')

media

readonly_fields = ('image_tag',)

class accounts.admin.**LaboratoryMembershipInLine**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

extra = 0

media

model
alias of accounts.models.laboratory_membership.LaboratoryMembership

class accounts.admin.**ProfileInline**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.StackedInline

can_delete = False

fields = ('title', 'image', 'image_tag', 'date_of_birth', 'institute', 'bio')

fk_name = 'user'

image_tag(*instance: accounts.models.profile.Profile*) → str

media

model
alias of *accounts.models.profile.Profile*

readonly_fields = ('image_tag',)

verbose_name_plural = 'Profile'

class accounts.admin.**UserAdmin**(*model*, *admin_site*)
Bases: django.contrib.auth.admin.UserAdmin

get_inline_instances(*request*, *obj=None*)

get_institute(*instance*)

inlines = (<class 'accounts.admin.ProfileInline'>, <class 'accounts.admin.LaboratoryMe

list_display = ('username', 'email', 'first_name', 'last_name', 'get_institute', 'is_s

list_select_related = ('profile',)

media

5.1.5 accounts.apps module

class accounts.apps.**AccountsConfig**(*app_name*, *app_module*)
Bases: django.apps.config.AppConfig

name = 'accounts'

ready()
Override this method in subclasses to run code when Django starts.

5.1.6 accounts.signals module

accounts.signals.**create_auth_token**(*sender*, *instance=None*, *created=False*, ***kwargs*)
Automatically creates an authentication token for the user, as described in the DRF documentation.

Parameters **created** (*bool, optional*) – Whether the instance is being created or updated, by default False

accounts.signals.**post_save_user_model_receiver**(*sender*, *instance*, *created*, **args*, ***kwargs*)

Standard implementation for user profile creation.

Parameters

- **sender** (*django.db.models.Model*) – The class of the instance being saved
- **instance** (*django.db.models.Model*) – The instance being saved
- **created** (*bool*) – Whether the instance is being created or updated

5.1.7 accounts.urls module

accounts.urls.**path**(*route*, *view*, *kwargs=None*, *name=None*, ***, *Pattern=<class 'django.urls.resolvers.RoutePattern'>*)

5.2 pylabber package

5.2.1 Module contents

5.2.2 Submodules

5.2.3 pylabber.settings module

Django settings for pylabber project.

For more information on this file, see https://docs.djangoproject.com/en/dev/topics/settings/

For the full list of settings and their values, see https://docs.djangoproject.com/en/dev/ref/settings/

pylabber.settings.**DICOM_IMPORT_MODE = 'minimal'**

Prevent the creation of DICOM data elements in the database.

pylabber.settings.**ENABLE_COUNT_FILTERING = False**

Do not annotate querysets with related model counts which drastically slows down queries.

5.2.4 pylabber.test_settings module

5.2.5 pylabber.urls module

pylabber URL Configuration

The urlpatterns list routes URLs to views.

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'))

`pylabber.urls.`**`path`**(*route*, *view*, *kwargs=None*, *name=None*, *, *Pattern=<class 'django.urls.resolvers.RoutePattern'>*)

5.2.6 pylabber.utils module

5.2.7 pylabber.wsgi module

WSGI config for pylabber project.

It exposes the WSGI callable as a module-level variable named `application`.

For more information on this file, see https://docs.djangoproject.com/en/2.0/howto/deployment/wsgi/

5.3 research package

5.3.1 Module contents

A reusable Django app to manage research information.

5.3.2 Subpackages

research.filters package

Module contents

Filters for *the app's models*.

Submodules

research.filters.subject_filter module

Definition of the *`SubjectFilter`* class.

`class` `research.filters.subject_filter.`**`SubjectFilter`**(*data=None*, *queryset=None*, *, *request=None*, *prefix=None*)

Bases: `django_filters.rest_framework.filterset.FilterSet`

Provides useful filtering options for the *`Subject`* model.

`base_filters = {'born_after_date': <django_filters.filters.DateFilter object>, 'born_`

`declared_filters = {'born_after_date': <django_filters.filters.DateFilter object>, 'b`

`filter_by_dicom_patient`(*queryset*, *name*, *value*)

Find the subject that represents a particular DICOM `Patient` instance.

Parameters

- **queryset** (*django.db.models.QuerySet*) – The *Subject* queryset to filter.
- **name** (*str*) – The name of the model field to filter on.
- **value** (*int*) – DICOM Patient ID.

filter_by_measurement(*queryset*, *name*, *value*)

filter_by_procedure(*queryset*, *name*, *value*)

filter_by_questionnaire_id(*queryset*, *name*, *value*)

filter_by_studies(*queryset*, *name*, *value*)

filter_by_study_group(*queryset*, *name*, *value*)

filter_nullable_charfield(*queryset*, *name*, *values*)

research.models package

Module contents

Definition of the app's models. For an illustration of the relationship between the different models, see the *Overview*.

Submodules

research.models.choices module

Subclasses of the ChoiceEnum class used to represent raw and human-readable values for choices within the *research.models* module.

class research.models.choices.**DominantHand**
Bases: pylabber.utils.utils.ChoiceEnum

An Enum representing supported dominant hand options.

A = 'Ambidextrous'

L = 'Left'

R = 'Right'

class research.models.choices.**Gender**
Bases: pylabber.utils.utils.ChoiceEnum

An Enum representing supported gender options.

CIS = 'Cisgender'

OTHER = 'Other'

TRANS = 'Transgender'

class research.models.choices.**Sex**
Bases: pylabber.utils.utils.ChoiceEnum

An Enum representing supported sex options.

F = 'Female'

M = 'Male'

U = 'Other'

research.models.group module

Definition of the *Group* model.

class research.models.group.**Group**(**args*, ***kwargs*)

Bases: django_extensions.db.models.TitleDescriptionModel, django_extensions.db.models.TimeStampedModel

Represents a unique study group (i.e. a grouping of subjects according to some experimental design in the context of a study).

mri_scan_set

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

study

The study associated with this experimental group.

research.models.study module

Definition of the *Study* model.

class research.models.study.**Study**(**args*, ***kwargs*)

Bases: django_extensions.db.models.TitleDescriptionModel, django_extensions.db.models.TimeStampedModel

Represents a single study in the database.

collaborators

Researchers collaborating on this study.

get_absolute_url()

Returns the canonical URL for this instance.

References

- get_absolute_url()

Returns URL

Return type str

group_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

image
An optional image to supplement the description.

n_subjects
Returns the number of subjects associated with this study.

Returns Number of subjects associated with this study

Return type int

procedures
The experimental procedures associated with this study.

query_associated_subjects() → django.db.models.query.QuerySet
Returns a queryset of subjects associated with this study.

See also:

- *subject_set()*

Returns Subjects associated with this study

Return type models.QuerySet

subject_set
Returns a queryset of subjects associated with this study.

See also:

- *query_associated_subjects()*

Returns Subjects associated with this study

Return type models.QuerySet

subjects
Subjects associated with this study. This field is currently not used, but kept because in the future it might be used for "caching" associated subjects to save queries.

research.models.subject module

Definition of the *Subject* model.

class research.models.subject.**Subject**(**args*, ***kwargs*)
Bases: `django_extensions.db.models.TimeStampedModel`

Represents a single research subject. Any associated data model should be associated with this model.

BIDS_DIR_TEMPLATE = 'sub-{pk}'

build_bids_directory(*force: bool = False*, *persistent: bool = True*, *progressbar: bool = False*, *progressbar_position: int = 0*)

custom_attributes
Custom attributes dictionary.

date_of_birth
Subject's date of birth.

dominant_hand
Subject's dominant hand.

first_name
Subject's first name.

gender
Subject's gender.

get_absolute_url()
Returns the canonical URL for this instance.

References

- get_absolute_url()

Returns URL

Return type str

get_bids_directory() → pathlib.Path

get_dominant_hand_display(*, *field=<django.db.models.fields.CharField: dominant_hand>*)

get_full_name() → str
Returns a formatted string with the subject's full name (first name and then last name).

Returns Subject's full name

Return type str

get_gender_display(*, *field=<django.db.models.fields.CharField: gender>*)

get_personal_information() → pandas.core.series.Series
Temporary method to use an external table to retrieve subject personal information.

Returns Subject personal information

Return type pd.Series

get_questionnaire_data()
A method to link between a subject to it's questionnaire data.

Returns Subject and Questionnaire information.

Return type pd.Series

get_raw_information() → pandas.core.series.Series
Temporary method to use an external table to retrieve subject information.

Returns Subject information

Return type pd.Series

get_sex_display(*, *field=<django.db.models.fields.CharField: sex>*)

id_number
Some representative ID number unique to this subject.

last_name
Subject's last name.

mri_session_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

query_measurements() → django.db.models.query.QuerySet

query_procedures() → django.db.models.query.QuerySet

query_run_set() → django.db.models.query.QuerySet

query_scores(*analysis: Union[django_analyses.models.analysis.Analysis, Iterable[django_analyses.models.analysis.Analysis]] = None, analysis_title: Union[str, Iterable[str]] = None, analysis_version: Union[django_analyses.models.analysis_version.AnalysisVersion, Iterable[django_analyses.models.analysis_version.AnalysisVersion]] = None, analysis_version_title: Union[str, Iterable[str]] = None, atlas=None, atlas_title: Union[str, Iterable[str]] = None, metric=None, metric_title: Union[str, Iterable[str]] = None, region=None, region_title: Union[str, Iterable[str]] = None, region_index: Union[int, Iterable[int]] = None, hemisphere: str = None*) → django.db.models.query.QuerySet

query_studies() → django.db.models.query.QuerySet
Returns a queryset of *Study* instances this subject has data associated with.

Returns Associated studies

Return type models.QuerySet

query_study_groups() → django.db.models.query.QuerySet

save(**args*, ***kwargs*)
Overrides the model's `save()` method to process custom attributes.

Hint: For more information, see Django's documentation on overriding model methods.

---

sex
Subject's sex.

study_set
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

research.models.procedure module

Definition of the *Procedure* model.

class research.models.procedure.**Procedure**(**args*, ***kwargs*)
Bases: django_extensions.db.models.TitleDescriptionModel

Represents an experimental procedure.

add_event(*event: research.models.event.Event*, *index: int = None*)
Performs an event addition.

events
Represents an ordered list of events in a procedure.

get_absolute_url()
Returns the canonical URL for this instance.

References

- get_absolute_url()

Returns URL

Return type str

max_index
Returns the maximal `index` field value of any associated `ProcedureStep` instances. If there aren't any, returns -1.

Returns Maximal step index, or -1

Return type int

step_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

study_set
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

research.models.event module

Definition of the *Event* model.

class research.models.event.**Event**(**args*, ***kwargs*)
Bases: django_extensions.db.models.TitleDescriptionModel

Represents an event as a part of a procedure.

get_absolute_url()
Returns the canonical URL for this instance.

References

- get_absolute_url()

Returns URL

Return type str

measurementdefinition
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

procedure_set
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

procedurestep_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

task
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

research.models.task module

Definition of the *Task* model.

class research.models.task.**Task**(**args*, ***kwargs*)

Bases: *research.models.event.Event*

Represents an experimental task.

event_ptr

Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

event_ptr_id

get_absolute_url()

Returns the canonical URL for this instance.

References

- get_absolute_url()

Returns URL

Return type str

research.models.measurement_definition module

Definition of the *MeasurementDefinition* model.

class research.models.measurement_definition.**MeasurementDefinition**(**args*, ***kwargs*)

Bases: *research.models.event.Event*

Represents an experimental measurement definition.

content_type

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

content_type_id

event_ptr
Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

event_ptr_id

get_absolute_url()
Returns the canonical URL for this instance.

References

- get_absolute_url()

Returns URL

Return type str

get_instance_set() → django.db.models.query.QuerySet
Returns a queryset of collected measurements associated with this measurements definition.

Returns Collected data instances

Return type models.QuerySet

instance_set
Returns a queryset of collected measurements associated with this measurements definition.

Returns Collected data instances

Return type models.QuerySet

See also:

- *get_instance_set()*

mri_session_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

query_associated_studies() → django.db.models.query.QuerySet

research.models.validators module

Validators for django fields within the *research.models* module.

research.models.validators.**not_future**(*value*)

research.serializers package

Module contents

Django REST Framework serializers module for the *research* package.

Submodules

research.serializers.group module

Definition of the *GroupSerializer* class.

class research.serializers.group.**GroupReadSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: rest_framework.serializers.HyperlinkedModelSerializer

HyperlinkedModelSerializer for the *Group* model to be used in GET requests.

class research.serializers.group.**GroupSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: rest_framework.serializers.HyperlinkedModelSerializer

HyperlinkedModelSerializer for the *Group* model to be used in POST requests.

class research.serializers.group.**MiniStudySerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: rest_framework.serializers.ModelSerializer

A minimized *Study* ModelSerializer used in the *GroupReadSerializer* class.

research.serializers.study module

Definition of the *StudySerializer* class.

class research.serializers.study.**MiniStudySerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: rest_framework.serializers.HyperlinkedModelSerializer

Minified serializer for the *Study* model.

class research.serializers.study.**StudySerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: rest_framework.serializers.HyperlinkedModelSerializer

Serializer for the *Study* model.

research.serializers.subject module

Definition of the *SubjectSerializer* class.

class research.serializers.subject.**AdminSubjectSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: *research.serializers.subject.SubjectSerializer*

class research.serializers.subject.**SubjectSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: rest_framework.serializers.ModelSerializer

Base serializer for the *Subject* model.

research.views package

Module contents

Submodules

research.views.group module

Definition of the *GroupViewSet* class.

class research.views.group.**GroupViewSet**(***kwargs*)

Bases: pylabber.views.defaults.DefaultsMixin, rest_framework.viewsets.ModelViewSet

API endpoint that allows *Group* instances to be viewed or edited.

basename = None

description = None

detail = None

filter_class

alias of research.filters.group_filter.GroupFilter

get_serializer_class()

Return the class to use for the serializer. Defaults to using *self.serializer_class*.

You may want to override this if you need to provide different serializations depending on the incoming request.

(Eg. admins get full serialization, others get basic serialization)

name = None

queryset

suffix = None

research.views.study module

Definintion of the *StudyViewSet* class.

class research.views.study.**StudyViewSet**(***kwargs*)

Bases: pylabber.views.defaults.DefaultsMixin, rest_framework.viewsets.ModelViewSet

API endpoint that allows *Study* instances to be viewed or edited.

aggregate(*request*) → rest_framework.response.Response

Returns related model counts if count filtering is enabled.

Parameters **request** (*Request*) – API request

Returns Aggregated queryset or informational message

Return type Response

basename = None

description = None

detail = None

filter_class

alias of research.filters.study_filter.StudyFilter

filter_queryset(*queryset*)

Filters the returned study queryset according to the user's collaborations.

Parameters **queryset** (*QuerySet*) – Base *Study* queryset

Returns Studies in which the user is a collaborator

Return type QuerySet

get_queryset() → django.db.models.query.QuerySet

Overrides the parent *get_queryset()* method to apply aggregated annotation if count filtering is enabled.

Returns Patient queryset

Return type QuerySet

name = None

queryset

serializer_class

alias of *research.serializers.study.StudySerializer*

suffix = None

research.views.subject module

Definition of the *SubjectViewSet* class.

class research.views.subject.**SubjectViewSet**(***kwargs*)

Bases: pylabber.views.defaults.DefaultsMixin, rest_framework.viewsets.ModelViewSet

API endpoint that allows *Subject* instances to be viewed or edited.

basename = None

description = None

detail = None

export_files(*request*)

filter_class
alias of *research.filters.subject_filter.SubjectFilter*

filter_queryset(*queryset*)
Given a queryset, filter it with whichever filter backend is in use.

You are unlikely to want to override this method, although you may need to call it either from a list view, or from a custom *get_object* method if you want to apply the configured filtering backend to the default queryset.

get_serializer_class()
Return the class to use for the serializer. Defaults to using *self.serializer_class*.

You may want to override this if you need to provide different serializations depending on the incoming request.

(Eg. admins get full serialization, others get basic serialization)

name = None

ordering_fields = ('id', 'id_number', 'first_name', 'last_name', 'date_of_birth', 'sex

plot(*request*, **args*, ***kwargs*)

plot_script(*request*, **args*, ***kwargs*)

plot_summary(*request*, **args*, ***kwargs*)

queryset

suffix = None

to_csv(*request*, **args*, ***kwargs*)

5.3.3 Submodules

5.3.4 research.admin module

class research.admin.**CollaboratorsInline**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

class Media
Bases: object

css = {'all': ('research/css/hide_admin_original.css',)}

email(*instance*) → str

extra = 0

fields = ('user_id', 'title', 'first_name', 'last_name', 'username', 'email')

first_name(*instance*) → str

has_add_permission(*request*, *instance*)
Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

last_name(*instance*) → str

media

model
alias of research.models.study.Study_collaborators

readonly_fields = ('user_id', 'title', 'first_name', 'last_name', 'username', 'email')

title(*instance*) → str

username(*instance*) → str

verbose_name_plural = 'Collaborators'

class research.admin.**DecadeBornListFilter**(*request*, *params*, *model*, *model_admin*)
Bases: django.contrib.admin.filters.SimpleListFilter

DECADES = ('40s', '50s', '60s', '70s', '80s', '90s', '00s')

lookups(*request*, *model_admin*)
Returns a list of tuples. The first element in each tuple is the coded value for the option that will appear in the URL query. The second element is the human-readable name for the option that will appear in the right sidebar.

parameter_name = 'decade'

queryset(*request*, *queryset*)
Returns the filtered queryset based on the value provided in the query string and retrievable via *self.value()*.

title = 'decade born'

class research.admin.**GroupAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

inlines = (<class 'research.admin.ScanInline'>,)

list_display = ('id', 'study_', 'title', 'description', 'mri_scan_count')

list_filter = ('title', ('study__title', <class 'research.admin.custom_titled_filter.<

media

mri_scan_count(*instance: research.models.group.Group*) → int

readonly_fields = ('study_', 'mri_scan_count')

search_fields = ('study__title', 'title', 'description')

study_(*instance: research.models.group.Group*) → str

class research.admin.**MeasurementDefinitionAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

list_display = ('id', 'title', 'description', 'content_type', 'n_collected')

media

n_collected(*instance: research.models.measurement_definition.MeasurementDefinition*) → int

class research.admin.**ProcedureAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

inlines = (<class 'research.admin.StudyInline'>, <class 'research.admin.ProcedureStepI

list_display = ('id', 'title', 'description', 'step_count')

media

step_count(*instance: research.models.procedure.Procedure*) → int

class research.admin.**ProcedureInline**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

class Media
Bases: object

css = {'all': ('research/css/hide_admin_original.css',)}

can_delete = False

description(*instance*) → str

extra = 0

fields = ('id_link', 'title', 'description')

has_add_permission(*request*, *instance*)
Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

id_link(*instance*) → str

media

model
alias of research.models.study.Study_procedures

readonly_fields = ('id_link', 'title', 'description')

title(*instance*) → str

verbose_name_plural = 'Procedures'

class research.admin.**ProcedureStepInline**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

class Media
Bases: object

css = {'all': ('research/css/hide_admin_original.css',)}

can_delete = False

event_description(*instance: research.models.procedure_step.ProcedureStep*) → str

event_title(*instance: research.models.procedure_step.ProcedureStep*) → str

event_type(*instance: research.models.procedure_step.ProcedureStep*) → str

extra = 0

fields = ('index_link', 'event_type', 'event_title', 'event_description')

has_add_permission(*request*, *instance: research.models.procedure_step.ProcedureStep*)
Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

index_link(*instance: research.models.procedure_step.ProcedureStep*) → str

media

model
alias of research.models.procedure_step.ProcedureStep

readonly_fields = ('index_link', 'event_type', 'event_title', 'event_description')

verbose_name_plural = 'Steps'

class research.admin.**ScanInline**(*parent_model*, *admin_site*)

Bases: django_admin_inline_paginator.admin.TabularInlinePaginated

class Media

Bases: object

css = {'all': ('research/css/hide_admin_original.css',)}

comments(*instance*) → str

description(*instance*) → str

download(*instance*) → str

echo_time(*instance*) → float

extra = 0

fields = ('id_link', 'subject', 'session', 'number', 'time', 'description', 'echo_time

has_add_permission(*request*, *instance*)

Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

id_link(*instance*) → str

inversion_time(*instance*) → float

media

model

alias of django_mri.models.scan.Scan_study_groups

number(*instance*) → int

readonly_fields = ('id_link', 'subject', 'session', 'number', 'time', 'description', '

repetition_time(*instance*) → float

session(*instance*) → str

spatial_resolution(*instance*) → str

Returns a nicely formatted representation of the scan's spatial resolution.

Parameters **instance** (*django_mri.Scan_study_groups*) – Scan_study_groups instance

Returns Formatted spatial resolution representation

Return type str

subject(*instance*) → str

time(*instance*) → datetime.datetime

verbose_name_plural = 'Scans'

class research.admin.**SessionInLine**(*parent_model*, *admin_site*)

Bases: django.contrib.admin.options.TabularInline

class Media

Bases: object

css = {'all': ('research/css/hide_admin_original.css',)}

can_delete = False

download(*instance: django_mri.models.session.Session*) → str

extra = 0

fields = ('id_link', 'time', 'measurement', 'scan_count', 'comments', 'download')

has_add_permission(*request*, *instance: django_mri.models.session.Session*)
Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

id_link(*instance: django_mri.models.session.Session*) → str

media

model
alias of `django_mri.models.session.Session`

readonly_fields = ('id_link', 'time', 'scan_count', 'download')

scan_count(*instance: django_mri.models.session.Session*) → int

verbose_name_plural = 'MRI Sessions'

class research.admin.**StudyAdmin**(*model*, *admin_site*)
Bases: `django.contrib.admin.options.ModelAdmin`

exclude = ('subjects', 'collaborators', 'procedures')

fields = ('title', 'description', 'image', 'participant_list')

inlines = (<class 'research.admin.CollaboratorsInline'>, <class 'research.admin.Proced

list_display = ('title', 'description', 'created')

media

participant_list(*instance: research.models.study.Study*) → str

readonly_fields = ('participant_list',)

class research.admin.**StudyAssociationFilter**(*request*, *params*, *model*, *model_admin*)
Bases: `django.contrib.admin.filters.SimpleListFilter`

lookups(*request*, *model_admin*)
Must be overridden to return a list of tuples (value, verbose value)

parameter_name = 'study participation'

queryset(*request*, *queryset*)
Return the filtered queryset.

title = 'study participation'

class research.admin.**StudyInline**(*parent_model*, *admin_site*)
Bases: `django.contrib.admin.options.TabularInline`

class Media
Bases: `object`

css = {'all': ('research/css/hide_admin_original.css',)}

can_delete = False

description(*instance*) → str

extra = 0

fields = ('id_link', 'title', 'description')

has_add_permission(*request*, *instance: research.models.procedure_step.ProcedureStep*)
Return True if the given request has permission to add an object. Can be overridden by the user in subclasses.

id_link(*instance*) → str

media

model
alias of research.models.study.Study_procedures

readonly_fields = ('id_link', 'title', 'description')

title(*instance*) → str

verbose_name_plural = 'Studies'

class research.admin.**SubjectAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

actions = ('export_csv',)

changelist_view(*request*, *extra_context=None*)
The 'change list' admin view for this model.

export_csv(*request*, *queryset*)

inlines = (<class 'research.admin.SessionInLine'>,)

list_display = ('id', 'id_number', 'first_name', 'last_name', 'sex', 'date_of_birth',

list_filter = ('sex', 'dominant_hand', <class 'research.admin.DecadeBornListFilter'>,

media

n_mri_sessions(*instance: research.models.subject.Subject*) → int

readonly_fields = ('n_mri_sessions',)

search_fields = ('id', 'id_number', 'first_name', 'last_name', 'date_of_birth__year')

class research.admin.**SubjectsInline**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

date_of_birth(*instance*)

extra = 0

first_name(*instance*)

id_number(*instance*)

last_name(*instance*)

media

model
alias of research.models.study.Study_subjects

readonly_fields = ('id_number', 'first_name', 'last_name', 'sex', 'date_of_birth')

sex(*instance*)

verbose_name_plural = 'Subjects'

class research.admin.**TaskAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

list_display = ('id', 'title', 'description')

media

research.admin.**custom_titled_filter**(*title: str*)
Copied from SO: https://stackoverflow.com/a/21223908/4416932

5.3.5 research.apps module

class research.apps.**ResearchConfig**(*app_name*, *app_module*)
Bases: django.apps.config.AppConfig

name = 'research'

ready()
Loads the app's signals.

References

- ready()

5.3.6 research.urls module

research.urls.**path**(*route*, *view*, *kwargs=None*, *name=None*, *, *Pattern=<class 'django.urls.resolvers.RoutePattern'>*)

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

p

r

Index

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V