Pount

Release 0.0.1

Régis Witz

Sep 08, 2020

CONTENTS:

Pount is an elegant, simple and free software to help you structure, visualize and valorize your data.

CHAPTER

ONE

WHAT AM I READING?

Documentation for project **Pount**, which is available on Read the Docs.

CHAPTER

TWO

WHAT IS POUNT?

Scientific work is much more than just publications. Indeed, unpublished failures and experiments constitute a real wealth of untapped knowledge. Thus, to preserve and spread these research data helps to argue success, remember mistakes and reveal unexplored ideas. To store, reference and expose these data is a proof of sustainability and traceability, on which the scientific approach is based on, such as experiments reproducibility and extensibility.

In the open science context, **Pount** is a free, modular and interoperable ecosystem for accessing knowledge

CHAPTER

THREE

WHY SHOULD I USE IT?

Pount is ready for the needs of today's researchers.

- save and version your data (documents, images, videos, 3D models) ;
- structure your data according to your scientific field standards, and extend these standards with your very own custom metadata ;
- share your data with a unique identifier, a citation template, and set up dedicated permissions to your communities ;
- visualize your data with a fast and flawless rendering, and enhance them with contextual information and hyperlinks.

Need a long-term storage for your data? Check Zenodo, Dataverse or Nakala.

CHAPTER

FOUR

ARE THERE ANY RELATED REPOSITORIES?

Sources for Pount server / API are hosted here, whereas sources for Pount client / website are hosted here.

CHAPTER

FIVE

HOW CAN I TEST IT?

Just visit Pount website to discover its features first-hand.

If you're already convinced and want to deploy your very own instance of Pount, *this page* describes how to setup a working environment as a developer.

CHAPTER

SIX

HOW CAN I CONTRIBUTE?

If it helps you, just using Pount and telling people who might need it about it is fine.

If you want to be more involved, feel free to :

- submit bugs and feature requests, and to help validate them when they are committed ;
- review the documentation and help make sure everything is helpful and understandable ;
- make merge requests for anything from typos to new features ;
- … or anything else you have in mind !

CHAPTER

SEVEN

WHAT DOES "POUNT" MEAN?

Pount stands for *Plateforme OUverte Numérique Transdisciplinaire* in French. Pount is a reference to the Land of Punt (written *Pays de Pount* in French).

CHAPTER

EIGHT

IS IT ANY GOOD?

Yes.

CHAPTER

NINE

AND WHO ARE YOU?

Pount is supported by the following organisms:

- Maison Interuniversitaire des Sciences de l'Homme - Alsace (MISHA, UMR7044, USR3227)
- University of Strasbourg, and its Direction du Numérique (DNUM)

Feel free to contact pount@unistra.fr for any additional information, question or comment.

9.1 Contributing

First off, thanks for taking the time to contribute!

The following document is a set of guidelines for contributing to the different repositories hosted in the Pount group on Gitlab.

Remember that Pount purpose is to empower researchers, and make them able to make the best use of their data. However, research takes time and, truth is, doesn't have a lot of money. Thus, regarding the code of its components, Pount main value is *sustainability*. This include low-cost operation and easy maintenance.

What will this codebase become in 100 years ?
Answer is, we have no idea. But research data must remain available nevertheless. So by always enforcing lightweight, easy to replace, optional components, we think that we will always be able to manage every change that will happen.
And maybe even welcome it.

9.1.1 Unit testing

Unit test *everything* you implement. In other words, if you deliver anything that makes the number of statements uncovered by unit test *decrease*, you compromise the quality of the whole project. Don't do that. Don't postpone unit tests writing, you won't do it, and nobody will do it at your place. This is how legacy code is created. Unit test *now*, while the feature exact scope is still fresh in your mind.

Just to be sure, let's say it again:

- Never, **ever** merge code that decreases code coverage
- Improve code coverage whenever you can

Always push cleaner code that what you pulled.

9.1.2 Git commit messages

- Use the present tense: "*Add feature*" not "*Added feature*"
- Use the imperative mood: "*Move cursor to...*" not "*Moves cursor to...*"
- Limit the first line to 72 characters or less
- Reference issues and pull requests liberally after the first line
- Avoid purely aesthetic modifications, such as whitespace commits
- Consider starting the commit message with an applicable gitmoji. In particular:
  - `:sparkles:` when introducing new, finished, tested features
  - `:construction:` when commiting a well scoped requirement for a yet unfinished feature
  - `:recycle:` when refactoring code
  - `:fire:` when removing code or files
  - `:white_check_mark:` when updating tests, or improving test coverage
  - `:bug:` when fixing a bug
  - `:pencil:` when writing docs
  - `:globe_with_meridians:` when translating documentation or UI text
  - `:heavy_plus_sign:` when adding dependencies
  - `:arrow_up:` when upgrading dependencies
  - `:twisted_rightwards_arrows:` when merging branches
    Consider using fast forward commits instead.

    You can of course use the following kinds of commits. However, these should ALWAYS be squashed before being merged into `master` branch!
  - `:poop:` when introducing untested or unrefactored features
  - `:green_heart:` when fixing the CI build
  - `:rotating_light:` when removing linter warnings

9.1.3 Coding Styleguide

This is open source software. Consider the people who will read your code, and make it look nice for them.

All Python code must be compliant with PEP-8.
You can generally check your code compliancy with `tox -e pep8`.

All Javascript code is linted with ESLint.
You can generally check your code compliancy with `npm run lint`.

9.1.4 Dependencies management

Project Pount is licensed under the termes of the GNU Affero GPL licence. Thus, make sure you:

- Use only copyleft compatible licences ; here's how (here is the same in French).
- Use tools to help you double-check licenses of **all** your dependencies. Here is an example on how to do it for Node.js projects. For Python projects, you can use pkg_resources. This is valid for indirect dependencies, too!
- When you add a new dependency to a Pount project, *always document why*. Explain why you didn't choose the alternatives, and if there are any version issues. These explanations should always live at the same place as the dependencies. For Python projects, this means in the `requirements*.txt` files. For Node.js projects, this means in the `package.json` file. Talking about dependency listing files, you're responsible of them: don't let your IDE or builder or what have you modify them without noticing or even understanding what happens, and don't let their formatting be spoiled.

Remember that Pount repositories are meant to be usable by anyone, with the least possible hassle. Thus, keep modifications that are specific to you or your organization separated from the default repositories. These includes specific settings or urls, credentials or access methods, deployment scripts, Django migrations, and so on. Of course, unless otherwise stated you are free to fork our repositories to add those specifics, as long as you remain within the terms of the applicable license.

9.1.5 Documentation

Code documentation is generated by Sphinx, using reStructured Text syntax.
Do not use Markdown for Pount documentation. Here are some of the whys.

Language of code documentation and user documentation is English.

Document your code in a way that is the most helpful for your fellow developpers:

- If a class or function purpose and behaviour is not immediately understandable by reading its profile, by all means, document it.
- Always document external API functions.
- There's no need to rephrase what the function or class profile already says. You generally don't need to explain *what* happens ; instead, explain *why*, and *how* it happens. Add *details*, add *context*, add possible *issues*.
- Why is this necessary?
- Why does it work like it does, and not more trivially?
- What is the *purpose* of it? Consider providing a link to the specification of the concerned feature, and/or to the issue signaling the defect (especially in an external library).
- Are there some caveats or unspecified cases concerning its implementation? Be exhaustive documenting them. And, while you're at it, **unit test** them!
- How should it be used by the caller? Consider providing examples!
- Are there some brittle or difficult to understand parts in the implementation? If so, consider inserting comment into the body of the class/function to help update/fix tricky parts.

If you postpone code commenting, you won't do it. So do it. And do it *now*.

9.2 API setup

This document describes the procedure for installing the Pount *server* / API. The *client* / website installation is described here.

9.2.1 Install PostgreSQL

Install dependencies:

```
>>> sudo apt install postgresql postgresql-server-dev-10
```

Change to user `postgres`, login to postgresql prompt and (if necessary) set password:

```
>>> sudo -u postgres -i
>>> psql
>>> /password password
```

Create database, and check it is created with:

```
>>>  CREATE DATABASE pount-db;
>>> /l
```

Yeah, *there's an additional space* before `CREATE DATABASE` and other SQL commands ...

Alternative:

```
>>> psql
>>>  CREATE USER username;
>>> /du
>>> /q
>>> psql pount-db
>>>  GRANT ALL PRIVILEGES ON DATABASE pount-db TO username;
```

9.2.2 Install Elasticsearch

From the manual, you can chose the `.tar.gz` option. See this page for additional info.

Download sources (283Mo), **check** download is all good, then **extract** them in a subfolder:

```
>>> wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.6.2-
↪linux-x86_64.tar.gz
>>> wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.6.2-
↪linux-x86_64.tar.gz.sha512
>>> shasum -a 512 -c elasticsearch-7.6.2-linux-x86_64.tar.gz.sha512
>>> tar -xzf elasticsearch-7.6.2-linux-x86_64.tar.gz
>>> mv elasticsearch-7.6.2 elasticsearch
```

Run Elasticsearch:

```
>>> ./elasticsearch/bin/elasticsearch
```

To check everything is up and running, you can open a new terminal and run:

```
>>> curl -XGET http://localhost:9200
```

If you get some JSON answer like this one, you should be okay:

```
{
  "name" : "something",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "something something",
  "version" : {
    "number" : "5.6.16",
    "build_hash" : "something",
    "build_date" : "somewhere in the past",
    "build_snapshot" : false,
    "lucene_version" : "6.6.1"
  },
  "tagline" : "You Know, for Search"
}
```

9.2.3 Build Pount API

Get the sources:

```
>>> git clone git@gitlab.com:pount/server.git
>>> cd server
```

Setup your **virtual environment**:

```
>>> virtualenv -p /usr/bin/python3.8 venv
>>> source venv/bin/activate
```

You can learn why using a virtual environment is a good idea here, and more about `venv` here.

Alternatively, you can use virtualenv or pipenv, but `venv` is simple, shipped with python and is now the recommended way to do it, so this documentation will assume you use `venv`.

Download and **install dependencies**:

```
>>> pip install -r requirements/dev.txt
```

Build and **run unit tests**:

```
>>> tox
```

Run Pount server:

```
>>> python manage.py makemigrations
>>> python manage.py migrate
>>> python manage.py createsuperuser
>>> python manage.py runserver
```

9.3 latest

9.3.1 apps package

Subpackages

apps.api package

Subpackages

apps.api.importers package

Submodules

apps.api.importers.zenodo module

apps.api.importers.zenodo.**fetch**(*url*)

apps.api.importers.zenodo.**fetch_validate**(*url*, *errors*)

apps.api.importers.zenodo.**translate**(*response*)

Module contents

apps.api.models package

Submodules

apps.api.models.group module

class apps.api.models.group.**Group**(*id*, *name*, *group_ptr*, *settings*)

Bases: django.contrib.auth.models.Group

exception DoesNotExist

Bases: django.contrib.auth.models.Group.DoesNotExist

exception MultipleObjectsReturned

Bases: django.contrib.auth.models.Group.MultipleObjectsReturned

group_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.

group_ptr_id

settings

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

apps.api.models.init module

apps.api.models.init.**initialize_roles**()

apps.api.models.init.**initialize_templates**()

apps.api.models.item module

class apps.api.models.item.**Item**(*id*, *metadata*, *settings*, *set*, *owner*)

Bases: django.db.models.base.Model

exception DoesNotExist

Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: django.core.exceptions.MultipleObjectsReturned

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property is_public

metadata

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

owner

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.

owner_id

set

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.

set_id

settings

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property title

apps.api.models.permission module

class apps.api.models.permission.**Permission**(*id*, *user*, *role*, *set*)
Bases: django.db.models.base.Model

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

static create(*user*, *set*, *role_name*)

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

role
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.

role_id

set
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.

set_id

user
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.

user_id

class apps.api.models.permission.**Role**(*id*, *name*, *access_rights*)
Bases: django.db.models.base.Model

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

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

access_rights
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

name
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

permissions
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.

apps.api.models.set module

class `apps.api.models.set.`**Set**(*id*, *template*, *settings*, *owner*)
Bases: `django.db.models.base.Model`

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

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

get_role(*user*)

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property is_public

items
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.

objects = <django.db.models.manager.Manager object>

owner
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.

owner_id

permissions
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.

set_role(*user*, *role_name*)

settings
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

template
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.

template_id

static update_owner_permissions(*sender*, ***kwargs*)

apps.api.models.template module

class apps.api.models.template.**Template**(*id*, *metadata*, *settings*)
Bases: `django.db.models.base.Model`

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

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

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

metadata
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property name

objects = <django.db.models.manager.Manager object>

sets
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.

settings
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

apps.api.models.user module

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

Yeah, there's a built-in User model for authentication in Django. However it is "highly recommended" to use a custom user model, because updating this custom model is quite easy compared to updating the default User model which is quite a PITA. @see https://docs.djangoproject.com/en/3.0/topics/auth/customizing/

#using-a-custom-user-model-when-starting-a-project

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

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

get_next_by_date_joined(*, *field=<django.db.models.fields.DateTimeField: date_joined>*, *is_next=True*, ***kwargs*)

get_previous_by_date_joined(*, *field=<django.db.models.fields.DateTimeField: date_joined>*, *is_next=False*, ***kwargs*)

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.

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

info
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property is_creator

settings
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

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.

Module contents

class apps.api.models.**Group**(*id*, *name*, *group_ptr*, *settings*)
Bases: `django.contrib.auth.models.Group`

exception DoesNotExist
Bases: `django.contrib.auth.models.Group.DoesNotExist`

exception MultipleObjectsReturned
Bases: `django.contrib.auth.models.Group.MultipleObjectsReturned`

group_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.

group_ptr_id

settings
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class apps.api.models.**Item**(*id*, *metadata*, *settings*, *set*, *owner*)
Bases: `django.db.models.base.Model`

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property is_public

metadata
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

owner
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.

owner_id

set
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.

set_id

settings
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property title

class apps.api.models.**Permission**(*id*, *user*, *role*, *set*)
Bases: django.db.models.base.Model

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

static create(*user*, *set*, *role_name*)

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

role
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.

role_id

set
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.

set_id

user
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.

user_id

class apps.api.models.**Role**(*id*, *name*, *access_rights*)
Bases: `django.db.models.base.Model`

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

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

access_rights
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

name
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

permissions
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.

class apps.api.models.**Set**(*id*, *template*, *settings*, *owner*)
Bases: `django.db.models.base.Model`

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

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

get_role(*user*)

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property is_public

items
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.

objects = <django.db.models.manager.Manager object>

owner
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.

owner_id

permissions
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.

set_role(*user*, *role_name*)

settings
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

template
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.

template_id

static update_owner_permissions(*sender*, ***kwargs*)

class apps.api.models.**Template**(*id*, *metadata*, *settings*)
Bases: `django.db.models.base.Model`

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

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

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

metadata
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property name

objects = <django.db.models.manager.Manager object>

sets
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.

settings
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

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

Yeah, there's a built-in User model for authentication in Django. However it is "highly recommended" to use a custom user model, because updating this custom model is quite easy compared to updating the default User model which is quite a PITA. @see https://docs.djangoproject.com/en/3.0/topics/auth/customizing/

#using-a-custom-user-model-when-starting-a-project

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

get_next_by_date_joined(***, *field=<django.db.models.fields.DateTimeField: date_joined>*, *is_next=True*, ***kwargs*)

get_previous_by_date_joined(***, *field=<django.db.models.fields.DateTimeField: date_joined>*, *is_next=False*, ***kwargs*)

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.

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

info
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property is_creator

settings
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

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.

apps.api.serializers package

Submodules

apps.api.serializers.item module

class apps.api.serializers.item.**ItemSerializer**(**args*, ***kwargs*)
Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta
Bases: object

fields = ['url', 'metadata', 'settings', 'set', 'owner']

model
alias of *apps.api.models.item.Item*

validate_settings(*settings: dict*) → dict

validate_settings_source(*source: dict*, *errors: list*) → None
Custom validation method called by validate_settings.

apps.api.serializers.item.**get_importer**(*name: str*, *errors: list*)

apps.api.serializers.item.**validate_url**(*url: str*, *errors: list*) → None

apps.api.serializers.permission module

class apps.api.serializers.permission.**PermissionSerializer**(**args*, ***kwargs*)
Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta
Bases: object

fields = ['url', 'user', 'role', 'set']

model
alias of *apps.api.models.permission.Permission*

class apps.api.serializers.permission.**RoleSerializer**(**args*, ***kwargs*)
Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta
Bases: object

fields = ['url', 'name', 'access_rights']

model
alias of *apps.api.models.permission.Role*

apps.api.serializers.set module

class apps.api.serializers.set.**SetSerializer**(**args*, ***kwargs*)
Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta
Bases: object

fields = ['url', 'template', 'settings', 'items', 'permissions', 'owner']

model
alias of *apps.api.models.set.Set*

apps.api.serializers.template module

class apps.api.serializers.template.**TemplateSerializer**(**args*, ***kwargs*)
Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta
Bases: object

fields = ['url', 'metadata', 'settings']

model
alias of *apps.api.models.template.Template*

apps.api.serializers.user module

class apps.api.serializers.user.**UserSerializer**(**args*, ***kwargs*)
Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta
Bases: object

fields = ['url', 'username', 'info', 'settings']

model
alias of django.contrib.auth.models.User

Module contents

class apps.api.serializers.**ItemSerializer**(**args*, ***kwargs*)
Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta
Bases: object

fields = ['url', 'metadata', 'settings', 'set', 'owner']

model
alias of *apps.api.models.item.Item*

validate_settings(*settings: dict*) → dict

validate_settings_source(*source: dict*, *errors: list*) → None
Custom validation method called by validate_settings.

class apps.api.serializers.**PermissionSerializer**(**args*, ***kwargs*)
Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta
Bases: object

fields = ['url', 'user', 'role', 'set']

model
alias of *apps.api.models.permission.Permission*

class apps.api.serializers.**RoleSerializer**(**args*, ***kwargs*)
Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta
Bases: object

fields = ['url', 'name', 'access_rights']

model
alias of *apps.api.models.permission.Role*

class apps.api.serializers.**SetSerializer**(**args*, ***kwargs*)
Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta
Bases: object

fields = ['url', 'template', 'settings', 'items', 'permissions', 'owner']

model
alias of *apps.api.models.set.Set*

class apps.api.serializers.**TemplateSerializer**(**args*, ***kwargs*)
Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta
Bases: object

fields = ['url', 'metadata', 'settings']

model
alias of *apps.api.models.template.Template*

class apps.api.serializers.**UserSerializer**(**args*, ***kwargs*)
Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta
Bases: object

fields = ['url', 'username', 'info', 'settings']

model
alias of django.contrib.auth.models.User

apps.api.views package

Submodules

apps.api.views.item module

class apps.api.views.item.**ItemViewSet**(***kwargs*)
Bases: rest_framework.viewsets.ModelViewSet

API endpoint that allows items to be viewed or edited.

basename = None

description = None

detail = None

name = None

pagination_class
alias of *apps.api.pagination.StandardPagination*

perform_create(*serializer*)

queryset

serializer_class
alias of *apps.api.serializers.item.ItemSerializer*

suffix = None

apps.api.views.permission module

class apps.api.views.permission.**PermissionViewSet**(***kwargs*)
Bases: rest_framework.mixins.RetrieveModelMixin, rest_framework.viewsets.GenericViewSet

API endpoint that allows specific permissions to be retrieved, but nothing else (no create, no update, no delete).

basename = None

description = None

detail = None

name = None

pagination_class
alias of *apps.api.pagination.StandardPagination*

queryset

serializer_class
alias of *apps.api.serializers.permission.PermissionSerializer*

suffix = None

class apps.api.views.permission.**RoleViewSet**(***kwargs*)
Bases: rest_framework.mixins.RetrieveModelMixin, rest_framework.mixins.ListModelMixin, rest_framework.viewsets.GenericViewSet

API endpoint that allows roles to be viewed, but not edited.

basename = None

description = None

detail = None

name = None

pagination_class
alias of *apps.api.pagination.StandardPagination*

queryset

serializer_class
alias of *apps.api.serializers.permission.RoleSerializer*

suffix = None

apps.api.views.set module

class apps.api.views.set.**SetViewSet**(***kwargs*)
Bases: rest_framework.viewsets.ModelViewSet

API endpoint that allows sets to be viewed or edited.

basename = None

description = None

detail = None

name = None

pagination_class
alias of *apps.api.pagination.StandardPagination*

queryset

serializer_class
alias of *apps.api.serializers.set.SetSerializer*

suffix = None

apps.api.views.template module

class apps.api.views.template.**TemplateViewSet**(***kwargs*)
Bases: rest_framework.mixins.RetrieveModelMixin, rest_framework.mixins.ListModelMixin, rest_framework.viewsets.GenericViewSet

API endpoint that allows templates to be viewed, but not edited.

basename = None

description = None

detail = None

name = None

pagination_class
alias of *apps.api.pagination.StandardPagination*

queryset

serializer_class
alias of *apps.api.serializers.template.TemplateSerializer*

suffix = None

apps.api.views.user module

class apps.api.views.user.**UserViewSet**(***kwargs*)
Bases: rest_framework.mixins.RetrieveModelMixin, rest_framework.mixins.ListModelMixin, rest_framework.viewsets.GenericViewSet

API endpoint that allows users to be viewed, but not edited.

basename = None

description = None

detail = None

name = None

pagination_class
alias of *apps.api.pagination.StandardPagination*

queryset

serializer_class
alias of *apps.api.serializers.user.UserSerializer*

suffix = None

Module contents

class apps.api.views.**ItemViewSet**(***kwargs*)
Bases: rest_framework.viewsets.ModelViewSet

API endpoint that allows items to be viewed or edited.

basename = None

description = None

detail = None

name = None

pagination_class
alias of *apps.api.pagination.StandardPagination*

perform_create(*serializer*)

queryset

serializer_class
alias of *apps.api.serializers.item.ItemSerializer*

suffix = None

class apps.api.views.**PermissionViewSet**(***kwargs*)
Bases: rest_framework.mixins.RetrieveModelMixin, rest_framework.viewsets.GenericViewSet

API endpoint that allows specific permissions to be retrieved, but nothing else (no create, no update, no delete).

basename = None

description = None

detail = None

name = None

pagination_class
alias of *apps.api.pagination.StandardPagination*

queryset

serializer_class
alias of *apps.api.serializers.permission.PermissionSerializer*

suffix = None

class apps.api.views.**RoleViewSet**(***kwargs*)
Bases: rest_framework.mixins.RetrieveModelMixin, rest_framework.mixins.ListModelMixin, rest_framework.viewsets.GenericViewSet

API endpoint that allows roles to be viewed, but not edited.

basename = None

description = None

detail = None

name = None

pagination_class
alias of *apps.api.pagination.StandardPagination*

queryset

serializer_class
alias of *apps.api.serializers.permission.RoleSerializer*

suffix = None

class apps.api.views.**SetViewSet**(***kwargs*)
Bases: rest_framework.viewsets.ModelViewSet

API endpoint that allows sets to be viewed or edited.

basename = None

description = None

detail = None

name = None

pagination_class
alias of *apps.api.pagination.StandardPagination*

queryset

serializer_class
alias of *apps.api.serializers.set.SetSerializer*

suffix = None

class apps.api.views.**TemplateViewSet**(***kwargs*)
Bases: rest_framework.mixins.RetrieveModelMixin, rest_framework.mixins.ListModelMixin, rest_framework.viewsets.GenericViewSet

API endpoint that allows templates to be viewed, but not edited.

basename = None

description = None

detail = None

name = None

pagination_class
alias of *apps.api.pagination.StandardPagination*

queryset

serializer_class
alias of *apps.api.serializers.template.TemplateSerializer*

suffix = None

class apps.api.views.**UserViewSet**(***kwargs*)
Bases: rest_framework.mixins.RetrieveModelMixin, rest_framework.mixins.ListModelMixin, rest_framework.viewsets.GenericViewSet

API endpoint that allows users to be viewed, but not edited.

basename = None

description = None

detail = None

name = None

pagination_class
alias of *apps.api.pagination.StandardPagination*

queryset

serializer_class
alias of *apps.api.serializers.user.UserSerializer*

suffix = None

Submodules

apps.api.admin module

apps.api.apps module

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

name = 'api'

apps.api.pagination module

class apps.api.pagination.**CursorPagination**
Bases: rest_framework.pagination.CursorPagination

When using PagePagination, if user asks for page N, then an item is added somehow, and then user asks for page N+1, the last item of page N can be shown again as the first item in page N+1. This doesn't happen with CursorPagination, which keeps references to objects, and does not have to calculate content for each page.

ordering = '-updated'

page_size = 10

page_size_query_param = 'page_size'

class apps.api.pagination.**StandardPagination**
Bases: rest_framework.pagination.PageNumberPagination

max_page_size = 1000

page_size = 10

page_size_query_param = 'page_size'

apps.api.rules module

apps.api.rules.**initialize_rules**()

apps.api.urls module

Module contents

apps.search package

Submodules

apps.search.apps module

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

name = 'search'

apps.search.documents module

class apps.search.documents.**Item**(*meta=None*, ***kwargs*)
Bases: elasticsearch_dsl.document.Document

Item, as indexed by Elasticsearch

static create(*model*, *using=None*, *index=None*)
Factory method. Creates an Elasticsearch document from a Django Model. In other words, it converts an instance of *apps.api.models.Item* into an instance of *apps.search.documents.Item*.

Parameters

- **model** – *apps.api.models.Item* object to convert from.
- **using** – elasticsearch.Elasticsearch connection to use.
- **index** – Name of the index corresponding to the *apps.search.documents.Item* object to be created.

Returns New Elasticsearch document corresponding to model.

Return type *apps.search.documents.Item*

apps.search.search module

apps.search.search.**bulk_indexing**(*index*)

apps.search.search.**create_index**(*index*)

Module contents

apps.search.**bulk_indexing**(*index*)

apps.search.**create_index**(*index*)

Module contents

9.3.2 pount package

Subpackages

pount.middleware package

Submodules

pount.middleware.firebase module

Module contents

Submodules

pount.asgi module

ASGI config for pount project.

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

For more information on this file, see https://docs.djangoproject.com/en/3.0/howto/deployment/asgi/

pount.urls module

pount URL Configuration

The *urlpatterns* list routes URLs to views. For more information please see: https://docs.djangoproject.com/en/3.0/topics/http/urls/

Examples:

- Function views
  1. Add an import: from my_app import views
  2. Add a URL to urlpatterns: path(‘’, views.home, name=’home’)
- Class-based views
  1. Add an import: from other_app.views import Home
  2. Add a URL to urlpatterns: path(‘’, Home.as_view(), name=’home’)
- Including another URLconf
  1. Import the include() function: from django.urls import include, path
  2. Add a URL to urlpatterns: path(‘blog/’, include(‘blog.urls’))

pount.wsgi module

WSGI config for pount 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/3.0/howto/deployment/wsgi/

Module contents

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

p

INDEX

A

B

C

D

F

G

I

M

N

O

P

Q

R

S

T

U

V