django$_{c}onfig_{m}odelsDocumentation$

Release 2.7.0

edX Doc Team

Apr 03, 2024

CONTENTS

Configuration models for Django allowing config management with auditing.

Contents:

CHAPTER

ONE

DJANGO-CONFIG-MODELS

1.1

1.1.1 Purpose

This app allows other apps to easily define a configuration model that can be hooked into the admin site to allow configuration management with auditing.

1.1.2 Getting Started

Add `config_models` to your `INSTALLED_APPS` list.

1.1.3 Usage

Create a subclass of `ConfigurationModel`, with fields for each value that needs to be configured:

```
class MyConfiguration(ConfigurationModel):
    frobble_timeout = IntField(default=10)
    frazzle_target = TextField(defalut="debug")
```

This is a normal django model, so it must be synced and migrated as usual.

The default values for the fields in the `ConfigurationModel` will be used if no configuration has yet been created.

Register that class with the Admin site, using the `ConfigurationAdminModel`:

```
from django.contrib import admin

from config_models.admin import ConfigurationModelAdmin

admin.site.register(MyConfiguration, ConfigurationModelAdmin)
```

Use the configuration in your code:

```
def my_view(self, request):
    config = MyConfiguration.current()
    fire_the_missiles(config.frazzle_target, timeout=config.frobble_timeout)
```

Use the admin site to add new configuration entries. The most recently created entry is considered to be `current`.

1.1.4 Configuration

The current `ConfigurationModel` will be cached in the `configuration` django cache, or in the `default` cache if `configuration` doesn't exist. The `configuration` and `default` caches are specified in the django `CACHES` setting. The caching can be per-process, per-machine, per-cluster, or some other strategy, depending on the cache configuration.

You can specify the cache timeout in each `ConfigurationModel` by setting the `cache_timeout` property.

You can change the name of the cache key used by the `ConfigurationModel` by overriding the `cache_key_name` function.

1.1.5 Extension

`ConfigurationModels` are just django models, so they can be extended with new fields and migrated as usual. Newly added fields must have default values and should be nullable, so that rollbacks to old versions of configuration work correctly.

1.1.6 Documentation

The full documentation is at https://django-config-models.readthedocs.org.

1.1.7 License

The code in this repository is licensed under the AGPL 3.0 unless otherwise noted.

Please see `LICENSE.txt` for details.

1.1.8 Getting Help

If you're having trouble, we have discussion forums at discuss.openedx.org where you can connect with others in the community.

Our real-time conversations are on Slack. You can request a Slack invitation, then join our community Slack workspace.

For anything non-trivial, the best path is to open an issue in this repository with as many details about the issue you are facing as you can provide.

For more information about these options, see the Getting Help page.

1.1.9 How To Contribute

Contributions are very welcome.

Please read How To Contribute for details.

This project is currently accepting all types of contributions, bug fixes, security fixes, maintenance work, or new features. However, please make sure to have a discussion about your new feature idea with the maintainers prior to beginning development to maximize the chances of your change being accepted. You can start a conversation by creating a new issue on this repo summarizing your idea.

1.1.10 Open edX Code of Conduct

All community members are expected to follow the Open edX Code of Conduct.

1.1.11 People

The assigned maintainers for this component and other project details may be found in Backstage. Backstage pulls this data from the `catalog-info.yaml` file in this repo.

1.1.12 Reporting Security Issues

Please do not report security issues in public. Please email security@openedx.org.

CHAPTER

TWO

GETTING STARTED

If you have not already done so, create/activate a virtualenv. Unless otherwise stated, assume all terminal code below is executed within the virtualenv.

2.1 Install dependencies

Dependencies can be installed via the command below.

```
$ make requirements
```

CHAPTER

THREE

CONFIG_MODELS

3.1 config_models package

3.1.1 Subpackages

config_models.management package

Subpackages

config_models.management.commands package

Submodules

config_models.management.commands.populate_model module

Populates a ConfigurationModel by deserializing JSON data contained in a file.

class config_models.management.commands.populate_model.**Command**(*stdout=None*, *stderr=None*, *no_color=False*, *force_color=False*)

Bases: BaseCommand

This command will deserialize the JSON data in the supplied file to populate a ConfigurationModel. Note that this will add new entries to the model, but it will not delete any entries (ConfigurationModel entries are read-only).

add_arguments(*parser*)

Entry point for subclassed commands to add custom arguments.

handle(**args*, ***options*)

The actual logic of the command. Subclasses must implement this method.

```
help = '\n Populates a ConfigurationModel by deserializing the supplied JSON.\n\n
JSON should be in a file, with the following format:\n\n { "model":
"config_models.ExampleConfigurationModel",\n "data":\n [\n { "enabled":  True,\n
"color":  "black"\n ...\n },\n { "enabled":  False,\n "color":  "yellow"\n ...\n
},\n ...\n ]\n }\n\n A username corresponding to an existing user must be specified
to indicate who\n is executing the command.\n\n $ ... populate_model -f
path/to/file.json -u username\n '
```

Module contents

Module contents

3.1.2 Submodules

3.1.3 config_models.admin module

Admin site models for managing *ConfigurationModel* subclasses.

class config_models.admin.**ConfigurationModelAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

ModelAdmin for *ConfigurationModel* subclasses

add_view(*request*, *form_url=''*, *extra_context=None*)

change_view(*request*, *object_id*, *form_url=''*, *extra_context=None*)

date_hierarchy = 'change_date'

get_actions(*request*)

Get the actions.

get_displayable_field_names()

Return all field names, excluding reverse foreign key relationships.

get_list_display(*request*)

Get the list display.

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

Hook for specifying custom readonly fields.

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

Return True if the given request has permission to delete the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to delete the *obj* model instance. If *obj* is None, this should return True if the given request has permission to delete *any* object of the given type.

property media

revert(*request*, *queryset*)

Admin action to revert a configuration back to the selected value

save_model(*request*, *obj*, *form*, *change*)

Given a model instance save it to the database.

class config_models.admin.**KeyedConfigurationModelAdmin**(*model*, *admin_site*)

Bases: *ConfigurationModelAdmin*

ModelAdmin for *ConfigurationModel* subclasses that use extra keys (i.e. they have KEY_FIELDS set).

add_view(*request*, *form_url=''*, *extra_context=None*)

date_hierarchy = None

edit_link(*inst*)

Edit link for the change view

get_list_display(*request*)

Add a link to each row for creating a new row using the chosen row as a template

get_queryset(*request*)

Annote the queryset with an 'is_active' property that's true iff that row is the most recently added row for that particular set of KEY_FIELDS values. Filter the queryset to show only is_active rows by default.

list_filter = (<class 'config_models.admin.ShowHistoryFilter'>,)

property media

class config_models.admin.**ShowHistoryFilter**(*request*, *params*, *model*, *model_admin*)

Bases: ListFilter

Admin change view filter to show only the most recent (i.e. the "current") row for each unique key value.

choices(*changelist*)

Returns choices ready to be output in the template.

expected_parameters()

List the query string params used by this filter

has_output()

Should this filter be shown?

parameter_name = 'show_history'

queryset(*request*, *queryset*)

Filter the queryset. No-op since it's done by KeyedConfigurationModelAdmin

title = 'Status'

3.1.4 config_models.apps module

config_models Django application initialization.

class config_models.apps.**ConfigModelsConfig**(*app_name*, *app_module*)

Bases: AppConfig

Configuration for the config_models Django application.

name = 'config_models'

3.1.5 config_models.decorators module

Decorators for model-based configuration.

config_models.decorators.**require_config**(*config_model*)

View decorator that enables/disables a view based on configuration.

Arguments:

config_model (ConfigurationModel subclass): The class of the configuration model to check.

Returns:

HttpResponse: 404 if the configuration model is disabled,
otherwise returns the response from the decorated view.

3.1.6 config_models.models module

Django Model baseclass for database-backed configuration.

class config_models.models.**ConfigurationModel**(**args*, ***kwargs*)

Bases: Model

Abstract base class for model-based configuration

Properties:

cache_timeout (int): The number of seconds that this configuration
should be cached

KEY_FIELDS = ()

class Meta

Bases: object

abstract = False

ordering = ('-change_date',)

classmethod cache_key_name(**args*)

Return the name of the key to use to cache the current configuration

cache_timeout = 600

change_date

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

changed_by

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.

changed_by_id

classmethod current(**args*)

Return the active configuration entry, either from cache, from the database, or by creating a new empty entry (which is not persisted).

enabled

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

classmethod equal_to_current(*json*, *fields_to_ignore=('id', 'change_date', 'changed_by')*)

Compares for equality this instance to a model instance constructed from the supplied JSON. This will ignore any fields in *fields_to_ignore*.

Note that this method cannot handle fields with many-to-many associations, as those can only be set on a saved model instance (and saving the model instance will create a new entry). All many-to-many field entries will be removed before the equality comparison is done.

Args:

json: json representing an entry to compare fields_to_ignore: List of fields that should not be compared for equality. By default includes *id*, *change_date*, and *changed_by*.

Returns: True if the checked fields are all equivalent, else False

fields_equal(*instance*, *fields_to_ignore=('id', 'change_date', 'changed_by')*)

Compares this instance's fields to the supplied instance to test for equality. This will ignore any fields in *fields_to_ignore*.

Note that this method ignores many-to-many fields.

Args:

instance: the model instance to compare fields_to_ignore: List of fields that should not be compared for equality. By default includes *id*, *change_date*, and *changed_by*.

Returns: True if the checked fields are all equivalent, else False

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

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

classmethod is_enabled(**key_fields*)

Returns True if this feature is configured as enabled, else False.

Arguments:

key_fields: The positional arguments are the KEY_FIELDS used to identify the configuration to be checked.

classmethod key_values(**key_fields*, ***kwargs*)

Get the set of unique values in the configuration table for the given key[s]. Calling cls.current(*value) for each value in the resulting list should always produce an entry, though any such entry may have enabled=False.

Arguments:

key_fields: The positional arguments are the KEY_FIELDS to return. For example if you had a course embargo configuration where each entry was keyed on (country, course), then you might want to know "What countries have embargoes configured?" with cls.key_values('country'), or "Which courses have country restrictions?" with cls.key_values('course'). You can also leave this unspecified for the default, which returns the distinct combinations of all keys.

flat: If you pass flat=True as a kwarg, it has the same effect as in Django's 'values_list' method: Instead of returning a list of lists, you'll get one list of values. This makes sense to use whenever there is only one key being queried.

Return value:

List of lists of each combination of keys found in the database. e.g. [("Italy", "course-v1:SomeX+some+2015"), ...] for the course embargo example

classmethod key_values_cache_key_name(**key_fields*)

Key for fetching unique key values from the cache

objects

save(*force_insert=False*, *force_update=False*, *using=None*, *update_fields=None*)

Clear the cached value when saving a new configuration entry

class config_models.models.**ConfigurationModelManager**(**args*, ***kwargs*)

Bases: Manager

Query manager for ConfigurationModel

current_set()

A queryset for the active configuration entries only. Only useful if KEY_FIELDS is set.

Active means the means recent entries for each unique combination of keys. It does not necessaryily mean enbled.

with_active_flag()

A query set where each result is annotated with an 'is_active' field that indicates if it's the most recent entry for that combination of keys.

3.1.7 config_models.templatetags module

Override the submit_row template tag to remove all save buttons from the admin dashboard change view if the context has readonly marked in it.

config_models.templatetags.**submit_row**(*context*)

Overrides 'django.contrib.admin.templatetags.admin_modify.submit_row'.

Manipulates the context going into that function by hiding all of the buttons in the submit row if the key *readonly* is set in the context.

3.1.8 config_models.urls module

URLs for config_models.

3.1.9 config_models.utils module

Utilities for working with ConfigurationModels.

config_models.utils.**deserialize_json**(*stream*, *username*)

Given a stream containing JSON, deserializers the JSON into ConfigurationModel instances.

The stream is expected to be in the following format:

{ "model": "config_models.ExampleConfigurationModel",

"data":

[

{ "enabled": True,
"color": "black" ...

}, { "enabled": False,

"color": "yellow" ...

]

}

If the provided stream does not contain valid JSON for the ConfigurationModel specified, an Exception will be raised.

Arguments:

stream: The stream of JSON, as described above. username: The username of the user making the change. This must match an existing user.

Returns: the number of created entries

config_models.utils.**get_serializer_class**(*configuration_model*)

Returns a ConfigurationModel serializer class for the supplied configuration_model.

3.1.10 config_models.views module

API view to allow manipulation of configuration models.

class config_models.views.**AtomicMixin**

Bases: object

Mixin to provide atomic transaction for as_view.

classmethod as_view(***initkwargs*)

Overrides as_view to add atomic transaction.

classmethod create_atomic_wrapper(*wrapped_func*)

Returns a wrapped function.

class config_models.views.**ConfigurationModelCurrentAPIView**(***kwargs*)

Bases: *AtomicMixin*, CreateAPIView, RetrieveAPIView

This view allows an authenticated user with the appropriate model permissions to read and write the current configuration for the specified *model*.

Like other APIViews, you can use this by using a url pattern similar to the following:

```
url(r'config/example_config$', ConfigurationModelCurrentAPIView.as_
↪view(model=ExampleConfig))
```

authentication_classes = (<class 'rest_framework.authentication.SessionAuthentication'>,)

get_object()

Returns the object the view is displaying.

You may want to override this if you need to provide non-standard queryset lookups. Eg if objects are referenced using multiple keyword arguments in the url conf.

get_queryset()

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

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

(Eg. return a list of items that is specific to the user)

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)

model = None

perform_create(*serializer*)

permission_classes = (<class 'config_models.views.ReadableOnlyByAuthors'>,)

class config_models.views.**ReadableOnlyByAuthors**

Bases: DjangoModelPermissions

Only allow access by users with *add* permissions on the model.

perms_map = {'DELETE': ['%(app_label)s.delete_%(model_name)s'], 'GET': ['%(app_label)s.add_%(model_name)s'], 'HEAD': ['%(app_label)s.add_%(model_name)s'], 'OPTIONS': ['%(app_label)s.add_%(model_name)s'], 'PATCH': ['%(app_label)s.change_%(model_name)s'], 'POST': ['%(app_label)s.add_%(model_name)s'], 'PUT': ['%(app_label)s.change_%(model_name)s']}

3.1.11 Module contents

Configuration models for Django allowing config management with auditing.

CHAPTER

FOUR

TESTING

django_config_models has an assortment of test cases and code quality checks to catch potential problems during development. To run them all in the version of Python you chose for your virtualenv:

```
$ make validate
```

To run just the unit tests:

```
$ make test
```

To run just the code quality checks:

```
$ make quality
```

To run the unit tests under every supported Python version and the code quality checks:

```
$ make test-all
```

To generate and open an HTML report of how much of the code is covered by test cases:

```
$ make coverage
```

CHAPTER

FIVE

INTERNATIONALIZATION

All user-facing text content should be marked for translation. Even if this application is only run in English, our open source users may choose to use another language. Marking content for translation ensures our users have this choice.

Follow the internationalization coding guidelines in the edX Developer's Guide when developing new features.

5.1 Updating Translations

This project uses Transifex to translate content. After new features are developed the translation source files should be pushed to Transifex. Our translation community will translate the content, after which we can retrieve the translations.

Pushing source translation files to Transifex requires access to the edx-platform. Request access from the Open Source Team if you will be pushing translation files. You should also configure the Transifex client if you have not done so already.

The *make* targets listed below can be used to push or pull translations.

Target	Description
pull_translations	Pull translations from Transifex
push_translations	Push source translation files to Transifex

5.2 Fake Translations

As you develop features it may be helpful to know which strings have been marked for translation, and which are not. Use the *fake_translations* make target for this purpose. This target will extract all strings marked for translation, generate fake translations in the Esperanto (eo) language directory, and compile the translations.

You can trigger the display of the translations by setting your browser's language to Esperanto (eo), and navigating to a page on the site. Instead of plain English strings, you should see specially-accented English strings that look like this:

    Thé Fütüré øf Ønlïné Édüçätïøn # Før änýøné, änýwhéré, änýtïmé #

CHAPTER

SIX

CHANGE LOG

6.1 [2.6.0] - 2024-03-30

- Adding python3.11 and 3.12 support.

6.2 [2.5.1] - 2023-09-22

- Fixed issues with Django 4.2

6.3 [2.5.0] - 2023-08-07

- Fixed `ConfigurationModel.current`: it will make sure that it does not return None for current configuration.

6.4 [2.4.0] - 2023-07-19

- Added support for Django42 in CI
- Switch from `edx-sphinx-theme` to `sphinx-book-theme` since the former is deprecated

6.5 [2.3.0] - 2022-01-19

- Added Support for Django40 in CI
- Dropped Support for Django22, 30, 31

6.6 [2.2.2] - 2021-20-12

- Updated dependencies after removing unnecessary constraint on edx-django-utils, so the constraint will no longer be advertised.

6.7 [2.2.1] - 2021-20-12

- Replaced deprecated 'django.utils.translation.ugettext' with 'django.utils.translation.gettext'

6.8 [2.2.0] - 2021-07-14

- Added support for django3.2

6.9 [2.1.2] - 2021-06-24

- Move out django pin from base.in. Now it is coming from global constraint. Ran make upgrade.

6.10 [2.1.1] - 2021-01-28

- Fix deprecated reference of `util.memcache.safe_key`

6.11 [2.1.0] - 2021-01-12

- Dropped Python 3.5 Support

6.12 [2.0.2] - 2020-05-10

- Fix html escaping of edit links in admin

6.13 [2.0.1] - 2020-05-08

- Dropped support for Django<2.2
- Dropped support for python3.6
- Added support for python3.8

6.14 [2.0.0] - 2020-02-06

- Dropping support for Python 2.7
- Switch to using edx-django-utils TieredCache (a two-layer cache that uses both Django's cache and an internal request-level cache) to reduce the number of memcached roundtrips. This was a major performance issue that accounted for 10-20% of transaction time for certain courseware views in edx-platform.
- It is now REQUIRED to add *RequestCacheMiddleware* to middleware to use ConfigModels.
- Remove usage of the "configuration" cache setting. ConfigModels now always use the default Django cache.
- Django Rest Framework 3.7 and 3.8 are no longer supported.

6.15 [1.0.1] - 2019-04-23

- Fix auto publishing to PyPI

6.16 [1.0.0] - 2019-04-23

CHAPTER

SEVEN

CHANGED

- Unpin django-rest-framework requirements. This is a potentially **breaking change** if people were relying on this package to ensure the correct version of djangorestframework was being installed.

7.1 [0.2.0] - 2018-07-13

CHAPTER

EIGHT

ADDED

- Support for Python 3.6

CHAPTER

NINE

REMOVED

- Testing against Django 1.8 - 1.10

CHAPTER

TEN

CHANGED

- Updated dependency management to follow OEP-18

10.1 [0.1.10] - 2018-05-21

CHAPTER

ELEVEN

CHANGED

- Don’t assume the user model is Django’s default auth.User

11.1 [0.1.9] - 2017-08-07

CHAPTER

TWELVE

CHANGED

- Updated Django REST Framework dependency to 3.6 as we were not actually compatible with 3.2.

12.1 [0.1.8] - 2017-06-19

CHAPTER

THIRTEEN

ADDED

- Support for Django 1.11.

13.1 [0.1.7] - 2017-06-19

- Unreleased version number

13.2 [0.1.6] - 2017-06-01

CHAPTER

FOURTEEN

ADDED

- Support for Django 1.10.

14.1 [0.1.1] - [0.1.5] - 2017-06-01

14.1.1 Added

- Add quality testing to travis run.
- Add encrypted password for package PyPI.

CHAPTER

FIFTEEN

REMOVED

- Remove the quality condition on deployment.
- Remove the version combos known to fail.

CHAPTER

SIXTEEN

CHANGED

- Allow for lower versions of djangorestframework, to be compatible with edx-platform.
- Constrict DRF to version that works.
- Update versions of requirements via pip-compile.
- Use different test target - test-all instead of validate.

CHAPTER

SEVENTEEN

FIXED

- Fix name and supported versions.

17.1 [0.1.0] - 2016-10-06

17.1.1 Added

- First release on PyPI.

CHAPTER

EIGHTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

W