marsha Documentation

Stéphane "Twidi" Angel

May 23, 2018

Contents:

CHAPTER 1

About

A FUN video provider for Open edX

Hosted on Github

CHAPTER 2

ADR - Architecture Decision Records

2.1 Purpose

These are the architectural decisions that are taken while developing Marsha.

The format is based on Michael Nygard's article on the subject.

2.2 Proposed

2.3 Accepted

2.3.1 ADR 0001 - Actors

Status

Accepted

Context

There are different kinds of actors that need to interact with Marsha.

First we have the people managing a Marsha instance.

Then we have people linking their own website (this website is a "consumer site") to a Marsha instance, to host videos.

These consumer sites can host many publishers. We call these "organizations". And these organizations have managers that can administrate some things about authors, video sharing between authors...

Next we have the video authors, belonging to the organizations.

And finally we have the users coming to watch videos.

Decision

Let's separate those 5 actors / roles:

"staff"

Purpose

To manage a Marsha instance

Implementation

These are simply instances of the Django `User` model, with the flag `is_staff` set to `True`.

"admins"

Purpose

To manage the link between a consumer site and a Marsha instance, and the organizations allowed to access this consumer site on the instance.

Implementation

To represent a consumer site on a Marsha instance, we have a `ConsumerSite` Django model. With a `ManyToMany` link to the `User` model, named `admins` (not a single admin, to avoid having no admin if the only one existing is not available anymore)).

"managers"

Purpose

To manage the authors in the organization (an organization could be present on many consumer sites). To allow videos to be private to authors or public for all authors. And create courses.

Implementation

To represent an organization on a Marsha instance, we have an `Organization` Django model. With a `ManyToMany` link to the `User` model, named `managers`.

"authors"

Purpose

To post videos on a Marsha instance to be used on a consumer site.

Implementation

An author is simply an instance of the `User` model, but has a link to an `Organization` via a `ManyToMany` link, named `organizations` (we can imagine an author working for many organizations).

"viewers"

Purpose

To watch videos hosted on a Marsha instance.

Implementation

For the viewers we don't need to save anything in the database, so there is no instances of the `User` Django model for them.

Each time a user does an action to view a video, they access it via a url containing a unique token, with a limited life span. It's this token that grant them access to the video.

This is not the scope of this document to address token generations.

To store the user preferences regarding languages, video resolution, etc, it can simply be done via a cookie.

Consequences

Roles and authorizations are easy to understand.

The actors hierarchy is simple.

2.3.2 ADR 0002 - Videos languages

Status

Accepted

Context

We want to think Marsha as accessible from the beginning. At least from the point of view of the videos, which are the main content available.

We can think about a video as a main content, with many auxiliary contents.

Auxiliary contents

Audio

We have a main video, with an audio track included. The author could propose many other audio tracks, as audio files, and in the player the viewer can change the one to use.

Subtitles

In addition to audio tracks, many subtitles tracks can be available.

Sign language

Some people with disabilities could want a video with the sign language transcript. For this it can be a video incorporated in the original one, or an other video displayed on the site.

As sign languages are not the same for every spoken language, there can be several sign languages videos for a single video.

Decision

We decided to take all these elements into account right from the beginning.

So we have a main Django model named `Video`, from an author, with the link to to main video file, including the default audio track.

For the other audio tracks, we have `AudioTrack` Django model, with a `ForeignKey` to the `Video` instance, named `video`, and a `language` field (with only one audio track for each video+language)

It's the same for the subtitles, we have a `SubtitleTrack` Django model, with the same `video` and `language` fields, but with an additional `cc` field to indicate that this subtitle track is "closed captioning", ie a subtitles track for deaf or hard of hearing viewers. So there can be two subtitle tracks for each video+language: one with `cc` on, one with `cc` off.

And finally, for sign-languages videos, it's the same as for audio tracks: a Django model named `SignTrack` with the same `video` and `language` field.

Consequences

Accessibility is implemented from the start. Even if we decide to hide some things, the main concepts are here.

2.3.3 ADR 0003 - Content organization and accesses

Status

Accepted

Context

We have *actors*. And *videos*, with their auxiliary files, that we'll call for now "content". We want this content to be organized for actors to manage/view them.

Decision

Videos are grouped in playlists, which is a Django model named `Playlist`. A playlist belongs to an organization (`Organization` model defined in *actors*) and is created by someone who can be a manager of this organization, or an author who belongs to this organization. This link is materialized by an `author` field on the `Playlist` model, a `ForeignKey` to the `User` model.

The manager can allow many actors to manage a playlist, so there is a `ManyToMany` field on the `Playlist` model, named `editors`, pointing to the `User` model. And instead of relying on the hidden model created by Django when creating a `ManyToMany`, we'll define this model and use it via the `through` argument, to be able to add more rights constraints later if needed.

The author of the playlist is automatically added to this list of editors. And can be removed from it by a manager, still staying marked as the author, but being the author itself doesn't involve any rights.

A playlist can be duplicated by a manager, and if it stays in the same organization, the manager can clear or keep the list of editors. If it is to be duplicated in another organization, the list of editors will be cleared of actors not belonging to the new organization, and the manager will still be able to clear it all or keep the remaining editors.

When duplicated, a new instance of `Playlist` is created, with a link to the original playlist, keeping the author. We do the same for each instances of the `Video` linked to this playlist, but we will still point to the same files (videos/audios/subtitles...) on the hosting provider, to keep cost manageable.

And finally, there is a flag named `is_public` on the playlist, that can be toggled by a manager, to tell if the playlist can be viewed by anyone or only by people who were granted access to it. This kind of access is not in the scope of this document.

Consequences

Videos are grouped, which ease search and maintenance.

It is easy to change and check the rights for people to manage such a playlist.

2.3.4 ADR 0004 - Soft deletion

Status

Accepted

Context

We have users and objects, and everything is tied together. Deleting something may cause some problems, like deleting other things in cascade, or losing some relatioship, like not knowing who is the author of a video.

Decision

We don't want things to be deleted, instead we'll keep them in the database in a "deleted" state, so that they won't show up anywhere.

Looking at the ‘ Safe/Soft/Logical deletion/trashing and restoration/undeletion <https://djangopackages.org/grids/g/deletion/>‘_ page on djangopackages, we can make a choice with these constraints:

- support for python 3 and django 2
- simple, not over-featured
- can manage relationships
- supports the django admin
- is maintained

Regarding this, we choose django-safedelete which proposes many options to handle deletion, and so will fit our needs.

Consequences

If the correct deletion options are used, depending on the relationships, we won't lose data.

No new code to write to handle soft-deletion.

As a cons: one more dependency, but it seems maintained so it should be ok.

2.4 Deprecated

2.5 Superseded

CHAPTER 3

Development

At the time of writing, **Marsha** is developed with **Python** 3.6 for **Django 2.0**.

3.1 Code quality

We enforce good code by using some linters and auto code formatting tools.

To run all linters at once, run the command:

```
make lint
```

You can also run each linter one by one. We have ones for the following tools:

3.1.1 black

We use black to automatically format python files to produce pep 8 compliant code.

The best is to configure your editor to automatically update the files when saved.

If you want to do this manually, run the command:

```
make black
```

And to check if all is correct without actually modifying the files:

```
make check-black
```

3.1.2 flake8

In addition to *black* auto-formatting, we pass the code through flake8 to check for a lot of rules. In addition to the default `flake8` rules, we use these plugins:

- flake8-bugbear to find likely bugs and design problems.
- flake8-comprehensions to helps write better list/set/dict comprehensions.
- flake8-imports to check imports order via `isort`.
- flake8-mypy to check typing inconsistencies via `mypy` (see *mypy*).
- flake8-docstrings to check docstrings (see *Docstrings*)

To check your code, run the command:

```
make flake8
```

3.1.3 pylint

To enforce even more rules than the ones provided by *flake8*, we use pylint (with the help of pylint-django).

`pylint` may report some violations not found by `flake8`, and vice-versa. More often, both will report the same ones.

To check your code, run the command:

```
make pylint
```

3.1.4 mypy

We use python typing as much as possible, and mypy (with the help of mypy-django) to check it.

We also enforce it when defining Django fields, via the use of a "Django check". And the type of reverse related fields must always be defined.

To check if your code is valid for mypy, run the following command:

```
make mypy
```

Following is how the types of fields must be defined. To check if some fields typing is invalid, among other problems, run the following command:

```
make check-django
```

It will tell you all found errors in typing, with indication on how to correct them.

Scalar fields

For scalar fields (`CharField`, `IntegerField`, `BooleanField`, `DateField...`), we just add the type. For each type of Django field, there is an expected type. Theses types are defined in the `fields_type_mapping` dict defined in `marsha.core.base_models`. Add the missing ones if needed.

```
class Foo(models.Model):
    # we tell mypy that the attribute ``bar`` is of type ``str``
    name: str = models.CharField(...)
```

One-to-one fields

The type expected for a `OneToOneField` is the pointed model.

And on the pointed model we set the type of the related name to the source model.

```
class Foo(models.Model):
    # we tell mypy that the attribute ``bar`` is of type ``Bar``
    bar: "Bar" = models.OneToOneField(to=Bar, related_name="the_foo")

class Bar(models.Model):
    # we tell mypy that the class ``Bar`` has an attribute ``the_foo`` of type ``Foo``
    the_foo: Foo
```

Foreign keys

The type expected for a `ForeignKey` is the pointed model.

On the pointed model we have a many-to-one relationship. We use a type specifically defined for that, `ReverseFKType`, defined in `marsha.stubs`.

```
from marsha.stubs import ReverseFKType

class Foo(models.Model):
    # we tell mypy that the attribute ``bar`` is of type ``Bar``
    bar: "Bar" = models.ForeignKey(to=Bar, related_name="foos")

class Bar(models.Model):
    # we tell mypy that the class ``Bar`` has an attribute ``foos``
    # which is a reverse foreign key for the class ``Foo``
    foos: ReverseFKType[Foo]
```

Many-to-many fields

To define the type of a `ManyToManyField`, we use a type specifically defined for that, `M2MType`, defined in `marsha.stubs`.

On the pointed model, we use the same type, as it's also a many-to-many fields (ie it could have been defined in one model or the other).

```
from marsha.stubs import M2MType

class Foo(models.Model):
    # we tell mypy that the attribute ``bar`` is a many-to-many for the class ``Bar``
    bars: M2MType["Bar"] = models.ManyToManyField(to=Bar, related_name="foos")

class Bar(models.Model):
    # we tell mypy that the class ``Bar`` has an attribute ``foos``
    # which is a many-to-many for the class ``Foo``
    foos: M2MType[Foo]
```

3.2 Docstrings

flake8 is configured to enforce docstrings rule defined in the pep 257

In addition, we document function arguments, return types, etc… using the "NumPy" style documentation, which will be validated by *flake8*.

3.3 Django

3.3.1 Opinionated choices

We made the opinionated choice of following this document, "Tips for Building High-Quality Django Apps at Scale".

In particular:

- Do not split code in many Django applications if code is tightly coupled.
- Applications are inside the `marsha` package, not at root, so import are done like this:

```
from marsha.someapp.foo import bar
```

- Database tables are specifically named: we do not rely on the Django auto-generation. And then we don't prefix theses tables with the name of the project or the app. For example, a model named `Video`, will have the `db_table` attribute of its `Meta` class set to `video`. Enforced by a "Django check".
- Through tables for `ManyToManyField` relations must be defined. Enforced by a "Django check".

In addition:

- We enforce typing of fields and reverse related fields (see *mypy*). Enforced by a "Django check".
- We enforce defining a related name for all related field (`ManyToManyField`, `ForeignKey`, `OneToOneField`). Enforced by a "Django check".

To check if theses rules are correctly applied, among other rules defined by Django itself, run:

```
make check-django
```

Note: for these checks to work, all models must inherit from `BaseModel` defined in `marsha.core.base_models`.

3.3.2 Specific libraries

Here are a list of specific Django libraries we chose to use and why.

django-configurations

The aim is to be more specific about inheritance in settings from doc to staging to production, instead of relying on multiple files (and changing the `DJANGO_SETTINGS_MODULE` environment variable accordingly), using the `from .base import *` pattern.

It also provides tools to get some variables from the environment and validating them.

As a consequence of this tool, some default behavior of Django don't work anymore. It's why the `django-admin` bash command is redefined in `setup.cfg`.

django-safedelete

We don't want to lose data, so everything is kept in database, but hidden from users.

See *ADR 0004 - Soft deletion* for details about the reasoning behind this choice.

django-postgres-extra

With `django-safedelete`, model instances are not deleted but saved with a field `deleted` changing from `None` to the deletion date-time.

So we cannot anymore use `unique_together`.

`django-postgres-extra` provides a `ConditionalUniqueIndex` index, that acts like `unique_together`, but with a condition. We use the condition `WHERE "deleted" IS NULL`, to enforce the fact that only one non-deleted instance matching the fields combination can exist.

3.4 Tests

The whole Marsha project is tested.

Run this command to run all the tests:

```
make test
```

If you want to be more specific about the tests to run, use the Django command:

```
django-admin test marcha.path.to.module
django-admin test marcha.path.to.module.Class
django-admin test marcha.path.to.module.Class.method
```

3.5 Makefile

We provide a `Makefile` that allow to easily perform some actions.

make install Will install the project in the current environment, with its dependencies.

make dev Will install the project in the current environment, with its dependencies, including the ones needed in a development environment.

make dev-upgrade Will upgrade all default+dev dependencies defined in setup.cfg.

make check Will run all linters and checking tools.

make lint Will run all linters (*mypy*, *black*, *flake8*, *pylint*)

make mypy Will run the *mypy* tool.

make check-black Will run the *black* tool in check mode only (won't modify files)

make black Will run the *black* tool and update files that need to.

make flake8 Will run the *flake8* tool.

make pylint Will run the *pylint* tool.

make check-django Will run the Django `check` command.

make check-migrations Will check that all needed migrations exist.

make tests Will run django tests for the marsha project.

make doc Will build the documentation.

make dist Will build the package.

make clean Will clean python build related directories and files.

make full-clean Like `make clean` but will clean some other generated directories or files.

CHAPTER 4

Environment variables

We try to follow 12 factors app and so use environment variables for configuration.

Here is a list of the ones that are needed or optional:

4.1 DJANGO_SETTINGS_MODULE

Description Define the settings file to use

Type String

Mandatory Yes

Default None

Choices Must be set to `marsha.settings`

4.2 DJANGO_CONFIGURATION

Description Define the configuration to use in settings

Type String

Mandatory Yes

Default None

Choices Actually only `Dev` is available

4.3 DJANGO_SECRET_KEY

Description Used to provide cryptographic signing, and should be set to a unique, unpredictable value

Type String

Mandatory Yes

Default None

4.4 DJANGO_DEBUG

Description Turns on/off debug mode

Type Boolean

Mandatory No

Default `True` if `DJANGO_CONFIGURATION` is set to `Dev`, `False` otherwise

Choices `True` or `False`

4.5 DATABASE_URL

Description URL to represent the connection string to a database

Type String

Mandatory No if `DJANGO_CONFIGURATION` is set to `Dev`, yes otherwise

Default `sqlite:///path/to/project/db.sqlite3` if `DJANGO_CONFIGURATION` is set to `Dev`, None otherwise

Choices See schemas as presented by dj-database-url

CHAPTER 5

marsha

5.1 marsha package

5.1.1 Subpackages

marsha.core package

Subpackages

marsha.core.migrations package

Submodules

marsha.core.migrations.0001_initial_models module

class marsha.core.migrations.0001_initial_models.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('auth', '0009_alter_user_last_name_max_length')]

initial = True

operations = [<CreateModel name='User', fields=[('id', <django.db.models.fields.AutoFi

marsha.core.migrations.0002_soft_deletion module

class marsha.core.migrations.0002_soft_deletion.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('core', '0001_initial_models')]

operations = [<AlterModelManagers name='user', managers=[('objects', <marsha.core.mana

marsha.core.migrations.0003_missing_text_fields module

class marsha.core.migrations.0003_missing_text_fields.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [('core', '0002_soft_deletion')]

operations = [<AddField model_name='playlist', name='name', field=<django.db.models.fi

marsha.core.migrations.0004_duplicated_from__blank module

class marsha.core.migrations.0004_duplicated_from__blank.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [('core', '0003_missing_text_fields')]

operations = [<AlterField model_name='playlist', name='duplicated_from', field=<django

marsha.core.migrations.0005_use_our__nondeleteduniqueindex module

class marsha.core.migrations.0005_use_our__nondeleteduniqueindex.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [('core', '0004_duplicated_from__blank')]

operations = [<RemoveIndex model_name='audiotrack', name='audio_track_video_i_fe6276_i

Module contents

marsha.core.tests package

Submodules

marsha.core.tests.factories module

marsha.core.tests.test_models module

Module contents

Tests for the core app of the Marsha project.

Submodules

marsha.core.admin module

Admin of the core app of the Marsha project.

class marsha.core.admin.**AudioTrackInline**(*parent_model*, *admin_site*)
Bases: *marsha.core.admin.BaseTabularInline*

Inline for audio tracks of a video.

media

model
alias of *marsha.core.models.AudioTrack*

class marsha.core.admin.**AuthorOrganizationsInline**(*parent_model*, *admin_site*)
Bases: *marsha.core.admin.BaseTabularInline*

Inline for organizations the user is an author of.

media

model
alias of *marsha.core.models.Authoring*

verbose_name = 'authoring organization'

verbose_name_plural = 'authoring organizations'

class marsha.core.admin.**BaseModelAdmin**(*model*, *admin_site*)
Bases: safedelete.admin.SafeDeleteAdmin

Base for all our model admins.

media

class marsha.core.admin.**BaseTabularInline**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

Base for all our tabular inlines.

media

class marsha.core.admin.**ConsumerSiteAdmin**(*model*, *admin_site*)
Bases: *marsha.core.admin.BaseModelAdmin*

Admin class for the ConsumerSite model.

inlines = [<class 'marsha.core.admin.SiteAdminsInline'>, <class 'marsha.core.admin.Sit

list_display = ('name',)

media

class marsha.core.admin.**ManagedOrganizationsInline**(*parent_model*, *admin_site*)
Bases: *marsha.core.admin.BaseTabularInline*

Inline for organizations managed by a user.

media

model
alias of *marsha.core.models.OrganizationManager*

verbose_name = 'managed organization'

verbose_name_plural = 'managed organizations'

class marsha.core.admin.**MarshaAdminSite**(*name='admin'*)
Bases: django.contrib.admin.sites.AdminSite

Admin site for Marsha.

site_header = 'Marsha'

site_title = 'Marsha administration'

class marsha.core.admin.**OrganizationAdmin**(*model*, *admin_site*)
Bases: *marsha.core.admin.BaseModelAdmin*

Admin class for the Organization model.

inlines = [<class 'marsha.core.admin.OrganizationManagersInline'>, <class 'marsha.core

list_display = ('name',)

media

class marsha.core.admin.**OrganizationAuthorsInline**(*parent_model*, *admin_site*)
Bases: *marsha.core.admin.BaseTabularInline*

Inline for authors in an organization.

media

model
alias of *marsha.core.models.Authoring*

verbose_name = 'author'

verbose_name_plural = 'authors'

class marsha.core.admin.**OrganizationManagersInline**(*parent_model*, *admin_site*)
Bases: *marsha.core.admin.BaseTabularInline*

Inline for managers in an organization.

media

model
alias of *marsha.core.models.OrganizationManager*

verbose_name = 'manager'

verbose_name_plural = 'managers'

class marsha.core.admin.**OrganizationSitesInline**(*parent_model*, *admin_site*)
Bases: *marsha.core.admin.BaseTabularInline*

Inline for sites for an organization.

media

model
alias of *marsha.core.models.SiteOrganization*

verbose_name = 'site'

verbose_name_plural = 'sites'

class marsha.core.admin.**PlaylistAdmin**(*model*, *admin_site*)
Bases: *marsha.core.admin.BaseModelAdmin*

Admin class for the Playlist model.

exclude = ('duplicated_from',)

inlines = [<class 'marsha.core.admin.PlaylistVideosInline'>, <class 'marsha.core.admin

list_display = ('name', 'organization', 'author', 'is_public')

media

class marsha.core.admin.**PlaylistVideosInline**(*parent_model*, *admin_site*)
Bases: *marsha.core.admin.BaseTabularInline*

Inline for videos in a playlist.

media

model
alias of *marsha.core.models.PlaylistVideo*

verbose_name = 'video'

verbose_name_plural = 'videos'

class marsha.core.admin.**PlaystlistAccessesInline**(*parent_model*, *admin_site*)
Bases: *marsha.core.admin.BaseTabularInline*

Inline for with right to write access to a playlist.

media

model
alias of *marsha.core.models.PlaylistAccess*

verbose_name = 'user access'

verbose_name_plural = 'users accesses'

class marsha.core.admin.**SignTrackInline**(*parent_model*, *admin_site*)
Bases: *marsha.core.admin.BaseTabularInline*

Inline for sign tracks of a video.

media

model
alias of *marsha.core.models.SignTrack*

class marsha.core.admin.**SiteAdminsInline**(*parent_model*, *admin_site*)
Bases: *marsha.core.admin.BaseTabularInline*

Inline for admins of a site.

media

model
alias of *marsha.core.models.SiteAdmin*

verbose_name = 'admin'

verbose_name_plural = 'admins'

class marsha.core.admin.**SiteOrganizationsInline**(*parent_model*, *admin_site*)
Bases: *marsha.core.admin.BaseTabularInline*

Inline for organizations in a site.

media

model
alias of *marsha.core.models.SiteOrganization*

verbose_name = 'organization'

verbose_name_plural = 'organizations'

class marsha.core.admin.**SubtitleTrackInline**(*parent_model*, *admin_site*)
Bases: *marsha.core.admin.BaseTabularInline*

Inline for subtitle tracks of a video.

media

model
alias of *marsha.core.models.SubtitleTrack*

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

Admin class for the User model.

inlines = [<class 'marsha.core.admin.ManagedOrganizationsInline'>, <class 'marsha.core

media

class marsha.core.admin.**VideoAdmin**(*model*, *admin_site*)
Bases: *marsha.core.admin.BaseModelAdmin*

Admin class for the Video model.

exclude = ('duplicated_from',)

inlines = [<class 'marsha.core.admin.AudioTrackInline'>, <class 'marsha.core.admin.Sub

list_display = ('name', 'author', 'language')

media

marsha.core.apps module

Defines the django app config for the core app.

class marsha.core.apps.**CoreConfig**(*app_name*, *app_module*)
Bases: django.apps.config.AppConfig

Django app config for the core app.

name = 'marsha.core'

verbose_name = 'Marsha'

marsha.core.base_models module

Base models for the core app of the Marsha project.

class marsha.core.base_models.**BaseModel**(**args*, ***kwargs*)
Bases: safedelete.models.SafeDeleteModel

Base model for all our models.

It is based on SafeDeleteModel to easily manage how we want the instances to be deleted/soft-deleted, with or without its relationships. The default safedelete policy is SOFT_DELETE_CASCADE, ie the object to delete and its relations will be soft deleted: their deleted field will be filled with the current date-time (the opposite, None, is the same as "not deleted")

Also it adds some checks run with `django check`:

- check that all fields are correctly annotated.

- same for fields pointing to other models: final models must have all related

names properly annotated. - check that every `ManyToManyField` use a defined `through` table. - check that every model have a `db_table` defined, not prefixed with the name of the app or the project.

Parameters **deleted** (*DateTimeField*) – Deleted

classmethod check(***kwargs*) → List[django.core.checks.messages.CheckMessage]
Add checks for related names.

Parameters **kwargs** (*Any*) – Actually not used but asked by django to be present "for possible future usage".

Returns A list of the check messages representing problems found on the model.

Return type List[checks.CheckMessage]

validate_unique(*exclude: List[str] = None*) → None
Add validation for our `NonDeletedUniqueIndex` replacing `unique_together`.

For the parameters, see `django.db.models.base.Model.validate_unique`.

class marsha.core.base_models.**NonDeletedUniqueIndex**(*fields: Sequence*, *name: str = None*) → None
Bases: `psqlextra.indexes.conditional_unique_index.ConditionalUniqueIndex`

A special ConditionalUniqueIndex for non deleted objects.

__init__(*fields: Sequence*, *name: str = None*) → None
Override default init to pass our predefined condition.

For the parameters, see `ConditionalUniqueIndex.__init__`.

condition = '"deleted" IS NULL'

deconstruct()
Remove `condition` as an argument to be defined in migrations.

marsha.core.managers module

This module holds the managers for the marsha models.

class marsha.core.managers.**UserManager**(*queryset_class=None*, **args*, ***kwargs*)
Bases: `django.contrib.auth.models.UserManager`, `safedelete.managers.SafeDeleteManager`

Extends the default manager for users with the one for soft-deletion.

marsha.core.models module

This module holds the models for the marsha project.

class marsha.core.models.**AudioTrack**(**args*, ***kwargs*)
Bases: *marsha.core.models.BaseTrack*

Model representing an additional audio track for a video.

Parameters

- **id** (*AutoField*) – Id
- **deleted** (*DateTimeField*) – Deleted

- **video** (ForeignKey to *Video*) – video for which this track is
- **language** (*CharField*) – language of this track

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

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

get_language_display(*, *field=<django.db.models.fields.CharField: language>*)
Autogenerated: Shows the label of the `language`

id
Model field: ID

video
Model field: video, accesses the *Video* model.

class marsha.core.models.**Authoring**(**args*, ***kwargs*)
Bases: *marsha.core.base_models.BaseModel*

Model representing authors in an organization.

`through` model between `Organization.authors` and `User.authoring`.

Parameters

- **id** (*AutoField*) – Id
- **deleted** (*DateTimeField*) – Deleted
- **user** (ForeignKey to *User*) – user having authoring access in this organization
- **organization** (ForeignKey to *Organization*) – organization on which the user is an author

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

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

id
Model field: ID

organization
Model field: organization, accesses the *Organization* model.

organization_id
Model field: organization

user
Model field: author, accesses the *User* model.

user_id
Model field: author

class marsha.core.models.**BaseTrack**(**args*, ***kwargs*)
Bases: *marsha.core.base_models.BaseModel*

Base model for different kinds of tracks tied to a video.

Parameters

- **deleted** (*DateTimeField*) – Deleted

- **video** (ForeignKey to *Video*) – video for which this track is
- **language** (*CharField*) – language of this track

get_language_display(*, *field=<django.db.models.fields.CharField: language>*)
Autogenerated: Shows the label of the *language*

language
Model field: track language

video
Model field: video, accesses the *Video* model.

video_id
Model field: video

class marsha.core.models.**ConsumerSite**(**args*, ***kwargs*)
Bases: *marsha.core.base_models.BaseModel*

Model representing an external site with access to the Marsha instance.

Parameters

- **id** (*AutoField*) – Id
- **deleted** (*DateTimeField*) – Deleted
- **name** (*CharField*) – Name of the site
- **admins** (*ManyToManyField*) – users able to manage this site

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

admins
Model field: administrators, accesses the M2M *ConsumerSite* model.

id
Model field: ID

name
Model field: name

organizations
Model field: sites, accesses the M2M *Organization* model.

organizations_links
Model field: site, accesses the M2M *SiteOrganization* model.

sites_admins
Model field: site, accesses the M2M *SiteAdmin* model.

class marsha.core.models.**Organization**(**args*, ***kwargs*)
Bases: *marsha.core.base_models.BaseModel*

Model representing an organization to manage its playlists on one or many sites.

Parameters

- **id** (*AutoField*) – Id
- **deleted** (*DateTimeField*) – Deleted
- **name** (*CharField*) – name of the organization

- **sites** (*ManyToManyField*) – sites where this organization is present
- **managers** (*ManyToManyField*) – users able to manage this organization
- **authors** (*ManyToManyField*) – users able to manage playlists in this organization

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

authoring
Model field: organization, accesses the M2M *Authoring* model.

authors
Model field: authors, accesses the M2M *Organization* model.

id
Model field: ID

managers
Model field: managers, accesses the M2M *Organization* model.

managers_links
Model field: organization, accesses the M2M *OrganizationManager* model.

name
Model field: name

playlists
Model field: organization, accesses the M2M *Playlist* model.

sites
Model field: sites, accesses the M2M *Organization* model.

sites_links
Model field: organization, accesses the M2M *SiteOrganization* model.

class marsha.core.models.**OrganizationManager**(**args*, ***kwargs*)
Bases: *marsha.core.base_models.BaseModel*

Model representing managers of organizations.

`through` model between `Organization.managers` and `User.managed_organizations`.

Parameters

- **id** (*AutoField*) – Id
- **deleted** (*DateTimeField*) – Deleted
- **user** (ForeignKey to *User*) – user managing this organization
- **organization** (ForeignKey to *Organization*) – organization managed by this user

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

id
Model field: ID

organization
Model field: organization, accesses the *Organization* model.

organization_id
Model field: organization

user
Model field: manager, accesses the *User* model.

user_id
Model field: manager

class marsha.core.models.**Playlist**(**args*, ***kwargs*)
Bases: *marsha.core.base_models.BaseModel*

Model representing a playlist which is a list of videos.

Parameters

- **id** (*AutoField*) – Id
- **deleted** (*DateTimeField*) – Deleted
- **name** (*CharField*) – name of the playlist
- **organization** (ForeignKey to *Organization*) – Organization
- **author** (ForeignKey to *User*) – Author
- **is_public** (*BooleanField*) – if this playlist can be viewed without any access control
- **duplicated_from** (ForeignKey to *Playlist*) – original playlist this one was duplicated from
- **editors** (*ManyToManyField*) – users allowed to manage this playlist
- **videos** (*ManyToManyField*) – videos in this playlist

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

author
Model field: author, accesses the *User* model.

author_id
Model field: author

duplicated_from
Model field: duplicate from, accesses the *Playlist* model.

duplicated_from_id
Model field: duplicate from

duplicates
Model field: duplicate from, accesses the M2M *Playlist* model.

editors
Model field: editors, accesses the M2M *Playlist* model.

id
Model field: ID

is_public
Model field: is public

name
Model field: name

organization
Model field: organization, accesses the *Organization* model.

organization_id
Model field: organization

users_accesses
Model field: playlist, accesses the M2M *PlaylistAccess* model.

videos
Model field: videos, accesses the M2M *Playlist* model.

videos_links
Model field: playlist, accesses the M2M *PlaylistVideo* model.

class marsha.core.models.**PlaylistAccess**(**args*, ***kwargs*)
Bases: *marsha.core.base_models.BaseModel*

Model representing a user having right to manage a playlist.

`through` model between `Playlist.editors` and `User.managed_playlists`.

Parameters

- **id** (*AutoField*) – Id
- **deleted** (*DateTimeField*) – Deleted
- **user** (ForeignKey to *User*) – user having rights to manage this playlist
- **playlist** (ForeignKey to *Playlist*) – playlist the user has rights to manage

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

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

id
Model field: ID

playlist
Model field: playlist, accesses the *Playlist* model.

playlist_id
Model field: playlist

user
Model field: user, accesses the *User* model.

user_id
Model field: user

class marsha.core.models.**PlaylistVideo**(**args*, ***kwargs*)
Bases: *marsha.core.base_models.BaseModel*

Model representing a video in a playlist.

`through` model between `Playlist.videos` and `Video.playlists`.

Parameters

- **id** (*AutoField*) – Id
- **deleted** (*DateTimeField*) – Deleted
- **video** (ForeignKey to *Video*) – video contained in this playlist
- **playlist** (ForeignKey to *Playlist*) – playlist containing this video
- **order** (*PositiveIntegerField*) – video order in the playlist

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

id
Model field: ID

order
Model field: order

playlist
Model field: playlist, accesses the *Playlist* model.

playlist_id
Model field: playlist

video
Model field: video, accesses the *Video* model.

video_id
Model field: video

class marsha.core.models.**SignTrack**(**args*, ***kwargs*)
Bases: *marsha.core.models.BaseTrack*

Model representing a signs language track for a video.

Parameters

- **id** (*AutoField*) – Id
- **deleted** (*DateTimeField*) – Deleted
- **video** (ForeignKey to *Video*) – video for which this track is
- **language** (*CharField*) – language of this track

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

get_language_display(*, *field=<django.db.models.fields.CharField: language>*)
Autogenerated: Shows the label of the language

id
Model field: ID

video
Model field: video, accesses the *Video* model.

class marsha.core.models.**SiteAdmin**(**args*, ***kwargs*)
Bases: *marsha.core.base_models.BaseModel*

Model representing users with access to manage a site.

`through` model between `ConsumerSite.admins` and `User.administrated_sites`.

Parameters

- **id** (*AutoField*) – Id
- **deleted** (*DateTimeField*) – Deleted
- **user** (ForeignKey to *User*) – user with access to the site
- **site** (ForeignKey to *ConsumerSite*) – site to which the user has access

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

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

id
Model field: ID

site
Model field: site, accesses the *ConsumerSite* model.

site_id
Model field: site

user
Model field: user, accesses the *User* model.

user_id
Model field: user

class marsha.core.models.**SiteOrganization**(**args*, ***kwargs*)
Bases: *marsha.core.base_models.BaseModel*

Model representing organizations in sites.

`through` model between `Organization.sites` and `ConsumerSite.organizations`.

Parameters

- **id** (*AutoField*) – Id
- **deleted** (*DateTimeField*) – Deleted
- **site** (ForeignKey to *ConsumerSite*) – site having this organization
- **organization** (ForeignKey to *Organization*) – organization in this site

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

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

id
Model field: ID

organization
Model field: organization, accesses the *Organization* model.

organization_id
Model field: organization

site
Model field: site, accesses the *ConsumerSite* model.

site_id
Model field: site

class marsha.core.models.**SubtitleTrack**(**args*, ***kwargs*)
Bases: *marsha.core.models.BaseTrack*

Model representing a subtitle track for a video.

Parameters

- **id** (*AutoField*) – Id
- **deleted** (*DateTimeField*) – Deleted
- **video** (ForeignKey to *Video*) – video for which this track is
- **language** (*CharField*) – language of this track
- **has_closed_captioning** (*BooleanField*) – if closed captioning (for death or hard of hearing viewers) is on for this subtitle track

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

get_language_display(**, field=<django.db.models.fields.CharField: language>*)
Autogenerated: Shows the label of the language

has_closed_captioning
Model field: closed captioning

id
Model field: ID

video
Model field: video, accesses the *Video* model.

class marsha.core.models.**User**(**args*, ***kwargs*)
Bases: *marsha.core.base_models.BaseModel*, django.contrib.auth.models.AbstractUser

Model representing a user that can be authenticated to act on the Marsha instance.

Parameters

- **id** (*AutoField*) – Id
- **password** (*CharField*) – Password
- **last_login** (*DateTimeField*) – Last login
- **is_superuser** (*BooleanField*) – Designates that this user has all permissions without explicitly assigning them.
- **username** (*CharField*) – Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.
- **first_name** (*CharField*) – First name

- **last_name** (*CharField*) – Last name
- **email** (*EmailField*) – Email address
- **is_staff** (*BooleanField*) – Designates whether the user can log into this admin site.
- **is_active** (*BooleanField*) – Designates whether this user should be treated as active. Unselect this instead of deleting accounts.
- **date_joined** (*DateTimeField*) – Date joined
- **deleted** (*DateTimeField*) – Deleted
- **groups** (*ManyToManyField*) – The groups this user belongs to. A user will get all permissions granted to each of their groups.
- **user_permissions** (*ManyToManyField*) – Specific permissions for this user.

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

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

administrated_sites
Model field: administrators, accesses the M2M *ConsumerSite* model.

author_organizations
Model field: authors, accesses the M2M *Organization* model.

authored_videos
Model field: author, accesses the M2M *Video* model.

authoring
Model field: author, accesses the M2M *Authoring* model.

created_playlists
Model field: author, accesses the M2M *Playlist* model.

get_next_by_date_joined(*, *field=<django.db.models.fields.DateTimeField: date_joined>*, *is_next=True*, ***kwargs*)
Autogenerated: Finds next instance based on `date_joined`.

get_previous_by_date_joined(*, *field=<django.db.models.fields.DateTimeField: date_joined>*, *is_next=False*, ***kwargs*)
Autogenerated: Finds previous instance based on `date_joined`.

groups
Model field: groups, accesses the M2M *User* model.

id
Model field: ID

logentry_set
Model field: user, accesses the M2M `LogEntry` model.

managed_organizations
Model field: managers, accesses the M2M *Organization* model.

managed_organizations_links
Model field: manager, accesses the M2M *OrganizationManager* model.

managed_playlists
Model field: editors, accesses the M2M *Playlist* model.

objects = <marsha.core.managers.UserManager object>

playlists_accesses
Model field: user, accesses the M2M *PlaylistAccess* model.

sites_admins
Model field: user, accesses the M2M *SiteAdmin* model.

user_permissions
Model field: user permissions, accesses the M2M *User* model.

class marsha.core.models.**Video**(**args*, ***kwargs*)
Bases: *marsha.core.base_models.BaseModel*

Model representing a video, by an author.

Parameters

- **id** (*AutoField*) – Id
- **deleted** (*DateTimeField*) – Deleted
- **name** (*CharField*) – name of the video
- **description** (*TextField*) – description of the video
- **author** (ForeignKey to *User*) – author of the video
- **language** (*CharField*) – language of the video
- **duplicated_from** (ForeignKey to *Video*) – original video this one was duplicated from

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

audiotracks
Model field: video, accesses the M2M *AudioTrack* model.

author
Model field: author, accesses the *User* model.

author_id
Model field: author

description
Model field: description

duplicated_from
Model field: duplicate from, accesses the *Video* model.

duplicated_from_id
Model field: duplicate from

duplicates
Model field: duplicate from, accesses the M2M *Video* model.

get_language_display(*, *field=<django.db.models.fields.CharField: language>*)
Autogenerated: Shows the label of the *language*

id
Model field: ID

language
Model field: language

name
Model field: name

playlists
Model field: videos, accesses the M2M *Playlist* model.

playlists_links
Model field: video, accesses the M2M *PlaylistVideo* model.

signtracks
Model field: video, accesses the M2M *SignTrack* model.

subtitletracks
Model field: video, accesses the M2M *SubtitleTrack* model.

marsha.core.views module

Views of the `core` app of the Marsha project.

Module contents

The `core` app of the Marsha project.

5.1.2 Submodules

5.1.3 marsha.settings module

Django settings for marsha project.

Uses django-configurations to manage environments inheritance and the loading of some config from the environment

class marsha.settings.**Base**
Bases: configurations.base.Configuration

Base configuration every configuration (aka environment) should inherit from.

It depends on an environment variable that SHOULD be defined: - DJANGO_SECRET_KEY

You may also want to override default configuration by setting the following environment variables: - DJANGO_DEBUG - DATABASE_URL

ABSOLUTE_URL_OVERRIDES = {}

ADMINS = []

ALLOWED_HOSTS = []

APPEND_SLASH = True

AUTHENTICATION_BACKENDS = ['django.contrib.auth.backends.ModelBackend']

AUTH_PASSWORD_VALIDATORS = [{'NAME': 'django.contrib.auth.password_validation.UserAttr

AUTH_USER_MODEL = 'core.User'

BASE_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/marsha/envs/develop/lib/p

CACHES = {'default': {'BACKEND': 'django.core.cache.backends.locmem.LocMemCache'}}

```
CACHE_MIDDLEWARE_ALIAS = 'default'
CACHE_MIDDLEWARE_KEY_PREFIX = ''
CACHE_MIDDLEWARE_SECONDS = 600
CSRF_COOKIE_AGE = 31449600
CSRF_COOKIE_DOMAIN = None
CSRF_COOKIE_HTTPONLY = False
CSRF_COOKIE_NAME = 'csrftoken'
CSRF_COOKIE_PATH = '/'
CSRF_COOKIE_SECURE = False
CSRF_FAILURE_VIEW = 'django.views.csrf.csrf_failure'
CSRF_HEADER_NAME = 'HTTP_X_CSRFTOKEN'
CSRF_TRUSTED_ORIGINS = []
CSRF_USE_SESSIONS = False
DATABASES = {}
DATABASE_ROUTERS = []
DATA_UPLOAD_MAX_MEMORY_SIZE = 2621440
DATA_UPLOAD_MAX_NUMBER_FIELDS = 1000
DATETIME_FORMAT = 'N j, Y, P'
DATETIME_INPUT_FORMATS = ['%Y-%m-%d %H:%M:%S', '%Y-%m-%d %H:%M:%S.%f', '%Y-%m-%d %H:%M
DATE_FORMAT = 'N j, Y'
DATE_INPUT_FORMATS = ['%Y-%m-%d', '%m/%d/%Y', '%m/%d/%y', '%b %d %Y', '%b %d, %Y', '%d
DEBUG = False
DEBUG_PROPAGATE_EXCEPTIONS = False
DECIMAL_SEPARATOR = '.'
DEFAULT_CHARSET = 'utf-8'
DEFAULT_CONTENT_TYPE = 'text/html'
DEFAULT_EXCEPTION_REPORTER_FILTER = 'django.views.debug.SafeExceptionReporterFilter'
DEFAULT_FILE_STORAGE = 'django.core.files.storage.FileSystemStorage'
DEFAULT_FROM_EMAIL = 'webmaster@localhost'
DEFAULT_INDEX_TABLESPACE = ''
DEFAULT_TABLESPACE = ''
DISALLOWED_USER_AGENTS = []
DOTENV_LOADED = None
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'localhost'
EMAIL_HOST_PASSWORD = ''
```

```
EMAIL_HOST_USER = ''
EMAIL_PORT = 25
EMAIL_SSL_CERTFILE = None
EMAIL_SSL_KEYFILE = None
EMAIL_SUBJECT_PREFIX = '[Django] '
EMAIL_TIMEOUT = None
EMAIL_USE_LOCALTIME = False
EMAIL_USE_SSL = False
EMAIL_USE_TLS = False
FILE_CHARSET = 'utf-8'
FILE_UPLOAD_DIRECTORY_PERMISSIONS = None
FILE_UPLOAD_HANDLERS = ['django.core.files.uploadhandler.MemoryFileUploadHandler', 'dj
FILE_UPLOAD_MAX_MEMORY_SIZE = 2621440
FILE_UPLOAD_PERMISSIONS = None
FILE_UPLOAD_TEMP_DIR = None
FIRST_DAY_OF_WEEK = 0
FIXTURE_DIRS = []
FORCE_SCRIPT_NAME = None
FORMAT_MODULE_PATH = None
FORM_RENDERER = 'django.forms.renderers.DjangoTemplates'
IGNORABLE_404_URLS = []
INSTALLED_APPS = ['django.contrib.admin.apps.SimpleAdminConfig', 'django.contrib.auth'
INTERNAL_IPS = []
LANGUAGES = [('en', 'english'), ('fr', 'french')]
LANGUAGES_BIDI = ['he', 'ar', 'fa', 'ur']
LANGUAGE_CODE = 'en-us'
LANGUAGE_COOKIE_AGE = None
LANGUAGE_COOKIE_DOMAIN = None
LANGUAGE_COOKIE_NAME = 'django_language'
LANGUAGE_COOKIE_PATH = '/'
LOCALE_PATHS = []
LOGGING = {}
LOGGING_CONFIG = 'logging.config.dictConfig'
LOGIN_REDIRECT_URL = '/accounts/profile/'
LOGIN_URL = '/accounts/login/'
LOGOUT_REDIRECT_URL = None
```

```
MANAGERS = []
MEDIA_ROOT = ''
MEDIA_URL = ''
MESSAGE_STORAGE = 'django.contrib.messages.storage.fallback.FallbackStorage'
MIDDLEWARE = ['django.middleware.security.SecurityMiddleware', 'django.contrib.session
MIGRATION_MODULES = {}
MONTH_DAY_FORMAT = 'F j'
NUMBER_GROUPING = 0
PASSWORD_HASHERS = ['django.contrib.auth.hashers.PBKDF2PasswordHasher', 'django.contri
PASSWORD_RESET_TIMEOUT_DAYS = 3
PREPEND_WWW = False
ROOT_URLCONF = 'marsha.urls'
SECRET_KEY = 'FooBar'
SECURE_BROWSER_XSS_FILTER = False
SECURE_CONTENT_TYPE_NOSNIFF = False
SECURE_HSTS_INCLUDE_SUBDOMAINS = False
SECURE_HSTS_PRELOAD = False
SECURE_HSTS_SECONDS = 0
SECURE_PROXY_SSL_HEADER = None
SECURE_REDIRECT_EXEMPT = []
SECURE_SSL_HOST = None
SECURE_SSL_REDIRECT = False
SERVER_EMAIL = 'root@localhost'
SESSION_CACHE_ALIAS = 'default'
SESSION_COOKIE_AGE = 1209600
SESSION_COOKIE_DOMAIN = None
SESSION_COOKIE_HTTPONLY = True
SESSION_COOKIE_NAME = 'sessionid'
SESSION_COOKIE_PATH = '/'
SESSION_COOKIE_SECURE = False
SESSION_ENGINE = 'django.contrib.sessions.backends.db'
SESSION_EXPIRE_AT_BROWSER_CLOSE = False
SESSION_FILE_PATH = None
SESSION_SAVE_EVERY_REQUEST = False
SESSION_SERIALIZER = 'django.contrib.sessions.serializers.JSONSerializer'
SHORT_DATETIME_FORMAT = 'm/d/Y P'
```

SHORT_DATE_FORMAT = 'm/d/Y'

SIGNING_BACKEND = 'django.core.signing.TimestampSigner'

SILENCED_SYSTEM_CHECKS = []

STATICFILES_DIRS = []

STATICFILES_FINDERS = ['django.contrib.staticfiles.finders.FileSystemFinder', 'django.

STATICFILES_STORAGE = 'django.contrib.staticfiles.storage.StaticFilesStorage'

STATIC_ROOT = None

STATIC_URL = '/static/'

TEMPLATES = [{'BACKEND': 'django.template.backends.django.DjangoTemplates', 'DIRS': []

TEST_NON_SERIALIZED_APPS = []

TEST_RUNNER = 'django.test.runner.DiscoverRunner'

THOUSAND_SEPARATOR = ','

TIME_FORMAT = 'P'

TIME_INPUT_FORMATS = ['%H:%M:%S', '%H:%M:%S.%f', '%H:%M']

TIME_ZONE = 'UTC'

USE_ETAGS = False

USE_I18N = True

USE_L10N = True

USE_THOUSAND_SEPARATOR = False

USE_TZ = True

USE_X_FORWARDED_HOST = False

USE_X_FORWARDED_PORT = False

WSGI_APPLICATION = 'marsha.wsgi.application'

X_FRAME_OPTIONS = 'SAMEORIGIN'

YEAR_MONTH_FORMAT = 'F Y'

class marsha.settings.**Dev**

Bases: *marsha.settings.Base*

Development environment settings.

We set `DEBUG` to `True` by default, configure the server to respond to all hosts, and use a local sqlite database by default.

ABSOLUTE_URL_OVERRIDES = {}

ADMINS = []

ALLOWED_HOSTS = ['*']

APPEND_SLASH = True

AUTHENTICATION_BACKENDS = ['django.contrib.auth.backends.ModelBackend']

AUTH_PASSWORD_VALIDATORS = [{'NAME': 'django.contrib.auth.password_validation.UserAttr

AUTH_USER_MODEL = 'core.User'

```
BASE_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/marsha/envs/develop/lib/p
CACHES = {'default':  {'BACKEND': 'django.core.cache.backends.locmem.LocMemCache'}}
CACHE_MIDDLEWARE_ALIAS = 'default'
CACHE_MIDDLEWARE_KEY_PREFIX = ''
CACHE_MIDDLEWARE_SECONDS = 600
CSRF_COOKIE_AGE = 31449600
CSRF_COOKIE_DOMAIN = None
CSRF_COOKIE_HTTPONLY = False
CSRF_COOKIE_NAME = 'csrftoken'
CSRF_COOKIE_PATH = '/'
CSRF_COOKIE_SECURE = False
CSRF_FAILURE_VIEW = 'django.views.csrf.csrf_failure'
CSRF_HEADER_NAME = 'HTTP_X_CSRFTOKEN'
CSRF_TRUSTED_ORIGINS = []
CSRF_USE_SESSIONS = False
DATABASES = {'default':  {'NAME': '/home/docs/checkouts/readthedocs.org/user_builds/ma
DATABASE_ROUTERS = []
DATA_UPLOAD_MAX_MEMORY_SIZE = 2621440
DATA_UPLOAD_MAX_NUMBER_FIELDS = 1000
DATETIME_FORMAT = 'N j, Y, P'
DATETIME_INPUT_FORMATS = ['%Y-%m-%d %H:%M:%S', '%Y-%m-%d %H:%M:%S.%f', '%Y-%m-%d %H:%M
DATE_FORMAT = 'N j, Y'
DATE_INPUT_FORMATS = ['%Y-%m-%d', '%m/%d/%Y', '%m/%d/%y', '%b %d %Y', '%b %d, %Y', '%d
DEBUG = False
DEBUG_PROPAGATE_EXCEPTIONS = False
DECIMAL_SEPARATOR = '.'
DEFAULT_CHARSET = 'utf-8'
DEFAULT_CONTENT_TYPE = 'text/html'
DEFAULT_EXCEPTION_REPORTER_FILTER = 'django.views.debug.SafeExceptionReporterFilter'
DEFAULT_FILE_STORAGE = 'django.core.files.storage.FileSystemStorage'
DEFAULT_FROM_EMAIL = 'webmaster@localhost'
DEFAULT_INDEX_TABLESPACE = ''
DEFAULT_TABLESPACE = ''
DISALLOWED_USER_AGENTS = []
DOTENV_LOADED = None
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
```

```
EMAIL_HOST = 'localhost'
EMAIL_HOST_PASSWORD = ''
EMAIL_HOST_USER = ''
EMAIL_PORT = 25
EMAIL_SSL_CERTFILE = None
EMAIL_SSL_KEYFILE = None
EMAIL_SUBJECT_PREFIX = '[Django] '
EMAIL_TIMEOUT = None
EMAIL_USE_LOCALTIME = False
EMAIL_USE_SSL = False
EMAIL_USE_TLS = False
FILE_CHARSET = 'utf-8'
FILE_UPLOAD_DIRECTORY_PERMISSIONS = None
FILE_UPLOAD_HANDLERS = ['django.core.files.uploadhandler.MemoryFileUploadHandler', 'dj
FILE_UPLOAD_MAX_MEMORY_SIZE = 2621440
FILE_UPLOAD_PERMISSIONS = None
FILE_UPLOAD_TEMP_DIR = None
FIRST_DAY_OF_WEEK = 0
FIXTURE_DIRS = []
FORCE_SCRIPT_NAME = None
FORMAT_MODULE_PATH = None
FORM_RENDERER = 'django.forms.renderers.DjangoTemplates'
IGNORABLE_404_URLS = []
INSTALLED_APPS = ['django.contrib.admin.apps.SimpleAdminConfig', 'django.contrib.auth'
INTERNAL_IPS = []
LANGUAGES = [('en', 'english'), ('fr', 'french')]
LANGUAGES_BIDI = ['he', 'ar', 'fa', 'ur']
LANGUAGE_CODE = 'en-us'
LANGUAGE_COOKIE_AGE = None
LANGUAGE_COOKIE_DOMAIN = None
LANGUAGE_COOKIE_NAME = 'django_language'
LANGUAGE_COOKIE_PATH = '/'
LOCALE_PATHS = []
LOGGING = {}
LOGGING_CONFIG = 'logging.config.dictConfig'
LOGIN_REDIRECT_URL = '/accounts/profile/'
```

```
LOGIN_URL = '/accounts/login/'
LOGOUT_REDIRECT_URL = None
MANAGERS = []
MEDIA_ROOT = ''
MEDIA_URL = ''
MESSAGE_STORAGE = 'django.contrib.messages.storage.fallback.FallbackStorage'
MIDDLEWARE = ['django.middleware.security.SecurityMiddleware', 'django.contrib.sessions
MIGRATION_MODULES = {}
MONTH_DAY_FORMAT = 'F j'
NUMBER_GROUPING = 0
PASSWORD_HASHERS = ['django.contrib.auth.hashers.PBKDF2PasswordHasher', 'django.contril
PASSWORD_RESET_TIMEOUT_DAYS = 3
PREPEND_WWW = False
ROOT_URLCONF = 'marsha.urls'
SECRET_KEY = 'FooBar'
SECURE_BROWSER_XSS_FILTER = False
SECURE_CONTENT_TYPE_NOSNIFF = False
SECURE_HSTS_INCLUDE_SUBDOMAINS = False
SECURE_HSTS_PRELOAD = False
SECURE_HSTS_SECONDS = 0
SECURE_PROXY_SSL_HEADER = None
SECURE_REDIRECT_EXEMPT = []
SECURE_SSL_HOST = None
SECURE_SSL_REDIRECT = False
SERVER_EMAIL = 'root@localhost'
SESSION_CACHE_ALIAS = 'default'
SESSION_COOKIE_AGE = 1209600
SESSION_COOKIE_DOMAIN = None
SESSION_COOKIE_HTTPONLY = True
SESSION_COOKIE_NAME = 'sessionid'
SESSION_COOKIE_PATH = '/'
SESSION_COOKIE_SECURE = False
SESSION_ENGINE = 'django.contrib.sessions.backends.db'
SESSION_EXPIRE_AT_BROWSER_CLOSE = False
SESSION_FILE_PATH = None
SESSION_SAVE_EVERY_REQUEST = False
```

```
SESSION_SERIALIZER = 'django.contrib.sessions.serializers.JSONSerializer'
SHORT_DATETIME_FORMAT = 'm/d/Y P'
SHORT_DATE_FORMAT = 'm/d/Y'
SIGNING_BACKEND = 'django.core.signing.TimestampSigner'
SILENCED_SYSTEM_CHECKS = []
STATICFILES_DIRS = []
STATICFILES_FINDERS = ['django.contrib.staticfiles.finders.FileSystemFinder', 'django.
STATICFILES_STORAGE = 'django.contrib.staticfiles.storage.StaticFilesStorage'
STATIC_ROOT = None
STATIC_URL = '/static/'
TEMPLATES = [{'BACKEND': 'django.template.backends.django.DjangoTemplates', 'DIRS': []
TEST_NON_SERIALIZED_APPS = []
TEST_RUNNER = 'django.test.runner.DiscoverRunner'
THOUSAND_SEPARATOR = ','
TIME_FORMAT = 'P'
TIME_INPUT_FORMATS = ['%H:%M:%S', '%H:%M:%S.%f', '%H:%M']
TIME_ZONE = 'UTC'
USE_ETAGS = False
USE_I18N = True
USE_L10N = True
USE_THOUSAND_SEPARATOR = False
USE_TZ = True
USE_X_FORWARDED_HOST = False
USE_X_FORWARDED_PORT = False
WSGI_APPLICATION = 'marsha.wsgi.application'
X_FRAME_OPTIONS = 'SAMEORIGIN'
YEAR_MONTH_FORMAT = 'F Y'
```

5.1.4 marsha.stubs module

Stubs for the whole project to be used for typing annotations.

class marsha.stubs.**M2MType**

Bases: typing.Generic

Stub to represent both sides of a django `ManyToManyField`.

add(**objs*) → None

clear() → None

create(***kwargs*) → T

get_or_create(***kwargs*) → Tuple[T, bool]

remove(**objs*) → None

set(*objs: Iterable[T], clear: Union[bool, NoneType] = False*) → None

update_or_create(***kwargs*) → Tuple[T, bool]

class marsha.stubs.**ReverseFKType**

Bases: typing.Generic

Stub to represent the related field of a django ForeignKey/OneToOneField.

add(**objs*, *bulk: Union[bool*, *NoneType] = True*) → None

clear(*bulk: bool = True*) → None

create(***kwargs*) → T

get_or_create(***kwargs*) → Tuple[T, bool]

remove(**objs*, *bulk: Union[bool*, *NoneType] = True*) → None

set(*objs: Iterable[T], bulk: Union[bool, NoneType] = True, clear: Union[bool, NoneType] = False*) → None

update_or_create(***kwargs*) → Tuple[T, bool]

marsha.stubs.**ReverseO2O**

alias of *marsha.stubs.ReverseFKType*

5.1.5 marsha.test_runner module

Provides a test-runner to avoid running django checks.

class marsha.test_runner.**NoCheckDiscoverRunner**(*pattern=None*, *top_level=None*, *verbosity=1*, *interactive=True*, *failfast=False*, *keepdb=False*, *reverse=False*, *debug_mode=False*, *debug_sql=False*, *parallel=0*, *tags=None*, *exclude_tags=None*, ***kwargs*)

Bases: django.test.runner.DiscoverRunner

A Django test runner that do not run checks.

run_checks() → None

Do not run checks.

5.1.6 marsha.urls module

Marsha URLs configuration.

5.1.7 marsha.wsgi module

WSGI script for the marsha project.

5.1.8 Module contents

Marsha project.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X

Y