PyWatch Documentation

Release 0.0.1

Lucas Simon Rodrigues Magalhaes

July 15, 2014

Contents

O PyWatch é um aplicativo que visa reunir as palestras, tutoriais e screencasts espalhados na internet. O objetivo principal é montar uma biblioteca sobre Django, Python e outros web frameworks.

Futuramente pretende-se utilizar web semântica para relacionar todo o tipo de conteúdo com seus respectivos autores.

A inspiração surgiu ao ver o projeto emberwatch.com.

CHAPTER 1

Contribuindo com o PyWatch

Como um projeto open source, o PyWatch da as boas vindas aos contribuintes de todas as formas

Exemplos para contribuir inclui:

- Códigos
- Melhorar a documentação
- Reportar bugs

CHAPTER 2

Executando um exemplo

Crie o ambiente virtual. Pode-se utilizar o virtualenv ou virtualenvwrapper. Fica a sua escolha.

```
cd ~/venvs
virtualenv pywatch
source pywatch/bin/activate
```

Baixe e instale o PyWatch.

```
git clone git@github.com:lucassimon/pywatch.com.br.git
cd pywatch.com.br
pip install -r requirements/dev.txt
```

Sincronize o banco de dados.

```
cd pywatch.com.br
python manage.py syncdb --migrate --settings=pywatch.settings.dev
```

Execute o PyWatch.

```
cd pywatch.com.br
python manage.py runserver --settings=pywatch.settings.dev
```

Contents:

2.1 Visão Geral

2.1.1 Dependências

- Django >= 1.6.1
- Fabric
- Jinja2
- MarkupSafe
- Pygments
- Python == 2.7.3
- South
- Sphinx

- Unipath
- argparse
- coverage
- dj-database-url
- django-appconf
- django-compressor
- django-debug-toolbar
- django-decouple
- django-discover-runner
- django-extensions
- django-filter
- django-nose
- djangorestframework
- docutils
- ecdsa
- flake8
- ipdb
- ipython
- mccabe
- mock
- model-mommy
- nose
- paramiko
- pep8
- pip
- psycopg2
- pycrypto
- pyflakes
- selenium
- setuptools
- six
- splinter
- sqlparse
- wsgiref
- yolk

2.1.2 Funcionalidades

- Escrito em Django
- CRUD de palestrantes
- API REST V1 palestrantes
- CRUD de palestras

2.1.3 Suporte ao Browser

Recomendamos o uso do Google Chrome ou Firefox para utilizar o sistema.

2.2 Começando

2.2.1 Buscando ajuda

Se você tiver problemas e não consegue descobrir como resolve-lo, você pode obter ajuda a partir de nosso sistema de tickets.

2.2.2 Configurando o ambiente.

Instalando os pacotes necessarios.

```
$ sudo aptitude install build-essential libpq-dev git git-core python-dev python-virtualenv python-p
```

2.2.3 Configurando o virtualenv ou virtualenvwrapper.

VirtualEnv.

```
$ mkdir ~/venvs
$ cd ~/venvs
$ virtualenv --unzip-setuptools pywatch
$ source ~/venvs/pywatch/bin/activate
```

Virtualenvwrapper.

```
$ mkdir ~/venvs
$ sudo pip install virtualenvwrapper
$ echo >> "source '/usr/local/bin/virtualenvwrapper.sh'" .bashrc/.zshrc
$ echo >> "WORKON_HOME='~/venvs'
$ mkvirtualenv pywatch
```

Ativar o ambiente.

```
$ workon pywatch
```

2.2.4 Clonando o repositorio do projeto

Definir uma pasta para conter os projetos: code, workspace-django, projetos etc...

```
$ mkdir ~/workspace-django
$ cd ~/workspace-django
$ git clone git@github.com:lucassimon/pywatch.com.br.git
$ cd pywatch.com.br
```

2.2.5 Instalando os pacotes do requirements.txt

Pacotes de desenvolvimento.

```
$ cd workspace-django/pywatch.com.br
$ pip install -r requirements/dev.txt
```

2.2.6 Executar o syncdb e fazer as migrações

Primeiro o syncdb.

```
$ cd pywatch.com.br
$ python manage.py syncdb --migrate --settings=pywatch.settings.dev
```

Segundo, executar as migrates.

```
$ python manage.py migrate --all --settings=pywatch.settings.dev
```

2.2.7 Executar o runserver

Execute.

```
$ cd pywatch.com.br
$ python manage.py runserver --settings=pywatch.settings.dev
```

2.3 Configurações no admin do django

Acessar.

```
localhost:8000/admin
```

Usuario: Definido no syncdb Senha: Definido no syncdb

2.4 Banco de dados

user: pywatch_db pass: pywatchforthewin

2.5 Como usar o sistema

Lorem Ipsum

2.6 Pacotes

Pacotes utilizados no sistema

2.6.1 Speakers/Palestrantes

App de palestrantes

Definições das models,admin,views e urls utilizados na app Speakers.

models – Speakers models

class speakers.models.**KindContact**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Classe para o contato

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

args

message

silent_variable_failure = True

KindContact.**KINDS = (('PH', u'Telefone'), ('E', u'E-mail'), ('FX', u'Fax'), ('FB', u'Facebook'), ('TT', u'Twitter'), ('C**

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

args

message

KindContact.**clean**()
Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

KindContact.**clean_fields**(*exclude=None*)
Cleans all fields and raises a ValidationError containing message_dict of all validation errors if any occur.

KindContact.**date_error_message**(*lookup_type*, *field*, *unique_for*)

KindContact.**delete**(*using=None*)

KindContact.**full_clean**(*exclude=None*)
Calls clean_fields, clean, and validate_unique, on the model, and raises a ValidationError for any errors that occured.

KindContact.**get_kind_display**(**moreargs*, ***morekwargs*)

KindContact.**kind = None**
Atributo da classe KindContact para escolher as opcoes setada na tupla KINDS

Caracteristicas: CharField max length: 2

KindContact.**objects = <django.db.models.manager.Manager object at 0x7f5bf7207b90>**

KindContact.**pk**

KindContact.**prepare_database_save**(*unused*)

KindContact.**save**(*force_insert=False*, *force_update=False*, *using=None*)
Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

KindContact.**save_base**(*raw=False*, *cls=None*, *origin=None*, *force_insert=False*, *force_update=False*, *using=None*)
Does the heavy-lifting involved in saving. Subclasses shouldn't need to override this method. It's separate from save() in order to hide the need for overrides of save() to pass around internal-only parameters ('raw', 'cls', and 'origin').

KindContact.**serializable_value**(*field_name*)
Returns the value of the field name for this instance. If the field is a foreign key, returns the id value, instead of the object. If there's no Field object with this name on the model, the model attribute's value is returned directly.

Used to serialize a field's value (in the serializer, or form output, for example). Normally, you would just access the attribute directly and not use this method.

KindContact.**speaker**
Atributo da classe KindContact para referenciar ao objeto da classe speaker

KindContact.**unique_error_message**(*model_class*, *unique_check*)

KindContact.**validate_unique**(*exclude=None*)
Checks unique constraints on the model and raises ValidationError if any failed.

KindContact.**value = None**
Atributo da classe KindContact para setar o valor da opção escolhida

Caracteristicas: CharField max length: 255

class speakers.models.**Speaker**(**args*, ***kwargs*)
Bases: core.models.TimeStampedModel

Classe model para criar um objeto model de palestrante.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

args

message

silent_variable_failure = True

class Speaker.**Meta**
Seta a classe como abstrata

abstract = False

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

args

message

Speaker.**bio = None**
Atributo da classe Speaker para setar a biografria do palestrante.

Caracteristicas: TextField

Speaker.**clean**()
Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

Speaker.**clean_fields**(*exclude=None*)
Cleans all fields and raises a ValidationError containing message_dict of all validation errors if any occur.

Speaker.**contacts**

Speaker.**date_error_message**(*lookup_type*, *field*, *unique_for*)

Speaker.**delete**(*using=None*)

Speaker.**full_clean**(*exclude=None*)
Calls clean_fields, clean, and validate_unique, on the model, and raises a `ValidationError` for any errors that occured.

Speaker.**get_absolute_url**(**moreargs*, ***morekwargs*)
Retorna o caminho absoluto da instancia do objeto, através do reverse usando namespace definido no arquivo urls.py

Speaker.**get_next_by_created**(**moreargs*, ***morekwargs*)

Speaker.**get_next_by_modified**(**moreargs*, ***morekwargs*)

Speaker.**get_previous_by_created**(**moreargs*, ***morekwargs*)

Speaker.**get_previous_by_modified**(**moreargs*, ***morekwargs*)

Speaker.**name = None**
Atributo da classe Speaker para setar o nome do palestrante.

Caracteristicas: max length: 255

Speaker.**objects = <speakers.managers.SpeakerMostRecentCreatedManager object at 0x7f5bf7207510>**

Speaker.**pk**

Speaker.**prepare_database_save**(*unused*)

Speaker.**save**(*force_insert=False*, *force_update=False*, *using=None*)
Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

Speaker.**save_base**(*raw=False*, *cls=None*, *origin=None*, *force_insert=False*, *force_update=False*, *using=None*)
Does the heavy-lifting involved in saving. Subclasses shouldn't need to override this method. It's separate from save() in order to hide the need for overrides of save() to pass around internal-only parameters ('raw', 'cls', and 'origin').

Speaker.**serializable_value**(*field_name*)
Returns the value of the field name for this instance. If the field is a foreign key, returns the id value, instead of the object. If there's no Field object with this name on the model, the model attribute's value is returned directly.

Used to serialize a field's value (in the serializer, or form output, for example). Normally, you would just access the attribute directly and not use this method.

Speaker.**slug = None**
Atributo da classe Speaker para setar o slug do palestrante.

Caracteristicas: max length: 255 unique: True

Speaker.**unique_error_message**(*model_class*, *unique_check*)

Speaker.**validate_unique**(*exclude=None*)
Checks unique constraints on the model and raises ValidationError if any failed.

admin – Speakers admin

class speakers.admin.**ContactInline**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

Formulario de contatos em linha

can_delete = True

declared_fieldsets

exclude = None

extra = 2

fields = None

fieldsets = None

filter_horizontal = ()

filter_vertical = ()

fk_name = None

form
alias of ModelForm

formfield_for_choice_field(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a database Field that has declared choices.

formfield_for_dbfield(*db_field*, ***kwargs*)
Hook for specifying the form Field instance for a given database Field instance.

If kwargs are given, they're passed to the form Field's constructor.

formfield_for_foreignkey(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ForeignKey.

formfield_for_manytomany(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ManyToManyField.

formfield_overrides = {}

formset
alias of BaseInlineFormSet

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

get_formset(*request*, *obj=None*, ***kwargs*)
Returns a BaseInlineFormSet class for use in admin add/change views.

get_ordering(*request*)
Hook for specifying field ordering.

get_prepopulated_fields(*request*, *obj=None*)
Hook for specifying custom prepopulated fields.

get_readonly_fields(*request*, *obj=None*)
Hook for specifying custom readonly fields.

has_add_permission(*request*)

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

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

lookup_allowed(*lookup*, *value*)

max_num = None

media

model
alias of `KindContact`

ordering = None

prepopulated_fields = {}

queryset(*request*)

radio_fields = {}

raw_id_fields = ()

readonly_fields = ()

template = 'admin/edit_inline/tabular.html'

verbose_name = None

verbose_name_plural = None

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

Classe admin utilizada no django admin para oferecer as opcoes de CRUD do model Speaker

action_checkbox(*obj*)
A list_display column containing a checkbox widget.

action_form
alias of `ActionForm`

actions = []

actions_on_bottom = False

actions_on_top = True

actions_selection_counter = True

add_form_template = None

add_view(**args*, ***kwargs*)
The 'add' admin view for this model.

change_form_template = None

change_list_template = None

change_view(**args*, ***kwargs*)
The 'change' admin view for this model.

changelist_view(**args*, ***kwargs*)
The 'change list' admin view for this model.

construct_change_message(*request*, *form*, *formsets*)
Construct a change message from a changed object.

date_hierarchy = 'created'

declared_fieldsets

delete_confirmation_template = None

delete_model(*request*, *obj*)
Given a model instance delete it from the database.

delete_selected_confirmation_template = None

delete_view(**args*, ***kwargs*)
The 'delete' admin view for this model.

exclude = None

fields = None

fieldsets = None

filter_horizontal = ()

filter_vertical = ()

form
alias of `ModelForm`

formfield_for_choice_field(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a database Field that has declared choices.

formfield_for_dbfield(*db_field*, ***kwargs*)
Hook for specifying the form Field instance for a given database Field instance.

If kwargs are given, they're passed to the form Field's constructor.

formfield_for_foreignkey(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ForeignKey.

formfield_for_manytomany(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ManyToManyField.

formfield_overrides = {}

get_action(*action*)
Return a given action from a parameter, which can either be a callable, or the name of a method on the ModelAdmin. Return is a tuple of (callable, name, description).

get_action_choices(*request, default_choices=[('', '———')]*)
Return a list of choices for use in a form object. Each choice is a tuple (name, description).

get_actions(*request*)
Return a dictionary mapping the names of all actions for this ModelAdmin to a tuple of (callable, name, description) for each action.

get_changelist(*request*, ***kwargs*)
Returns the ChangeList class for use on the changelist page.

get_changelist_form(*request*, ***kwargs*)
Returns a Form class for use in the Formset on the changelist page.

get_changelist_formset(*request*, ***kwargs*)
Returns a FormSet class for use on the changelist page if list_editable is used.

get_fieldsets(*request*, *obj=None*)
Hook for specifying fieldsets for the add form.

get_form(*request*, *obj=None*, ***kwargs*)
Returns a Form class for use in the admin add view. This is used by add_view and change_view.

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

get_inline_instances(*request*)

get_list_display(*request*)
Return a sequence containing the fields to be displayed on the changelist.

get_list_display_links(*request*, *list_display*)
Return a sequence containing the fields to be displayed as links on the changelist. The list_display parameter is the list of fields returned by get_list_display().

get_model_perms(*request*)
Returns a dict of all perms for this model. This dict has the keys `add`, `change`, and `delete` mapping to the True/False for each of those actions.

get_object(*request*, *object_id*)
Returns an instance matching the primary key provided. `None` is returned if no match is found (or the object_id failed validation against the primary key field).

get_ordering(*request*)
Hook for specifying field ordering.

get_paginator(*request*, *queryset*, *per_page*, *orphans=0*, *allow_empty_first_page=True*)

get_prepopulated_fields(*request*, *obj=None*)
Hook for specifying custom prepopulated fields.

get_readonly_fields(*request*, *obj=None*)
Hook for specifying custom readonly fields.

get_urls()

has_add_permission(*request*)
Returns True if the given request has permission to add an object. Can be overriden by the user in subclasses.

has_change_permission(*request*, *obj=None*)
Returns True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

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

has_delete_permission(*request*, *obj=None*)
Returns True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overriden 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.

history_view(*request*, *object_id*, *extra_context=None*)
The 'history' admin view for this model.

inlines = [<class 'speakers.admin.ContactInline'>]

list_display = ('name', 'slug', 'bio', 'created')

list_display_links = ()

list_editable = ()

list_filter = ('created',)

list_max_show_all = 200

list_per_page = 100

list_select_related = False

log_addition(*request*, *object*)
Log that an object has been successfully added.

The default implementation creates an admin LogEntry object.

log_change(*request*, *object*, *message*)
Log that an object has been successfully changed.

The default implementation creates an admin LogEntry object.

log_deletion(*request*, *object*, *object_repr*)
Log that an object will be deleted. Note that this method is called before the deletion.

The default implementation creates an admin LogEntry object.

lookup_allowed(*lookup*, *value*)

media

message_user(*request*, *message*)
Send a message to the user. The default implementation posts a message using the django.contrib.messages backend.

object_history_template = None

ordering = None

paginator
alias of `Paginator`

prepopulated_fields = {'slug': ('name',)}

queryset(*request*)
Returns a QuerySet of all model instances that can be edited by the admin site. This is used by changelist_view.

radio_fields = {}

raw_id_fields = ()

readonly_fields = ()

render_change_form(*request*, *context*, *add=False*, *change=False*, *form_url=''*, *obj=None*)

response_action(*request*, *queryset*)
Handle an admin action. This is called if a request is POSTed to the changelist; it returns an HttpResponse if the action was handled, and None otherwise.

response_add(*request*, *obj*, *post_url_continue='../%s/'*)
Determines the HttpResponse for the add_view stage.

response_change(*request*, *obj*)
Determines the HttpResponse for the change_view stage.

save_as = False

save_form(*request*, *form*, *change*)
Given a ModelForm return an unsaved instance. `change` is True if the object is being changed, and False if it's being added.

save_formset(*request*, *form*, *formset*, *change*)
Given an inline formset save it to the database.

save_model(*request*, *obj*, *form*, *change*)
Given a model instance save it to the database.

save_on_top = False

save_related(*request*, *form*, *formsets*, *change*)
Given the `HttpRequest`, the parent `ModelForm` instance, the list of inline formsets and a boolean value based on whether the parent is being added or changed, save the related objects to the database. Note that at this point save_form() and save_model() have already been called.

search_fields = ('name', 'slug', 'created')

urls

2.6.2 Talks/Palestras

App de palestras

Definições das models, views e urls utilizados na app Talks.

`models` – Talks models

class `talks.models.`**MediaTalk**(**args*, ***kwargs*)
Bases: `core.models.Media`

Model responsavel por criar as medias de palestrantes

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

args

message

silent_variable_failure = True

class `MediaTalk.`**Meta**
Seta a classe como abstrata

abstract = False

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

args

message

`MediaTalk.`**clean**()
Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

`MediaTalk.`**clean_fields**(*exclude=None*)
Cleans all fields and raises a ValidationError containing message_dict of all validation errors if any occur.

MediaTalk.**date_error_message**(*lookup_type*, *field*, *unique_for*)

MediaTalk.**delete**(*using=None*)

MediaTalk.**full_clean**(*exclude=None*)
Calls clean_fields, clean, and validate_unique, on the model, and raises a ValidationError for any errors that occured.

MediaTalk.**get_next_by_created**(**moreargs*, ***morekwargs*)

MediaTalk.**get_next_by_modified**(**moreargs*, ***morekwargs*)

MediaTalk.**get_previous_by_created**(**moreargs*, ***morekwargs*)

MediaTalk.**get_previous_by_modified**(**moreargs*, ***morekwargs*)

MediaTalk.**get_type_display**(**moreargs*, ***morekwargs*)

MediaTalk.**objects = <django.db.models.manager.Manager object at 0x7f5bf6fba550>**

MediaTalk.**pk**

MediaTalk.**prepare_database_save**(*unused*)

MediaTalk.**save**(*force_insert=False*, *force_update=False*, *using=None*)
Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

MediaTalk.**save_base**(*raw=False*, *cls=None*, *origin=None*, *force_insert=False*, *force_update=False*, *using=None*)
Does the heavy-lifting involved in saving. Subclasses shouldn't need to override this method. It's separate from save() in order to hide the need for overrides of save() to pass around internal-only parameters ('raw', 'cls', and 'origin').

MediaTalk.**serializable_value**(*field_name*)
Returns the value of the field name for this instance. If the field is a foreign key, returns the id value, instead of the object. If there's no Field object with this name on the model, the model attribute's value is returned directly.

Used to serialize a field's value (in the serializer, or form output, for example). Normally, you would just access the attribute directly and not use this method.

MediaTalk.**talk**
Atributo da classe MediaTalk para referenciar ao objeto da classe Talk

MediaTalk.**unique_error_message**(*model_class*, *unique_check*)

MediaTalk.**validate_unique**(*exclude=None*)
Checks unique constraints on the model and raises ValidationError if any failed.

class talks.models.**Talk**(**args*, ***kwargs*)
Bases: core.models.TimeStampedModel, core.models.StandardItemStuffModel

Model responsavel pelos palestrante

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

args

message

silent_variable_failure = True

class Talk.**Meta**
Seta a classe como abstrata

abstract = False

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

args

message

Talk.**clean**()
Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

Talk.**clean_fields**(*exclude=None*)
Cleans all fields and raises a ValidationError containing message_dict of all validation errors if any occur.

Talk.**date_error_message**(*lookup_type*, *field*, *unique_for*)

Talk.**delete**(*using=None*)

Talk.**full_clean**(*exclude=None*)
Calls clean_fields, clean, and validate_unique, on the model, and raises a ValidationError for any errors that occured.

Talk.**get_absolute_url**(**moreargs*, ***morekwargs*)
Retorna o caminho absoluto da instancia do objeto, através do reverse usando namespace definido no arquivo urls.py

Talk.**get_next_by_created**(**moreargs*, ***morekwargs*)

Talk.**get_next_by_modified**(**moreargs*, ***morekwargs*)

Talk.**get_previous_by_created**(**moreargs*, ***morekwargs*)

Talk.**get_previous_by_modified**(**moreargs*, ***morekwargs*)

Talk.**medias**

Talk.**objects = <talks.managers.TalkMostRecentCreatedManager object at 0x7f5bf6f98b10>**

Talk.**pk**

Talk.**prepare_database_save**(*unused*)

Talk.**save**(*force_insert=False*, *force_update=False*, *using=None*)
Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

Talk.**save_base**(*raw=False*, *cls=None*, *origin=None*, *force_insert=False*, *force_update=False*, *using=None*)
Does the heavy-lifting involved in saving. Subclasses shouldn't need to override this method. It's separate from save() in order to hide the need for overrides of save() to pass around internal-only parameters ('raw', 'cls', and 'origin').

Talk.**serializable_value**(*field_name*)
Returns the value of the field name for this instance. If the field is a foreign key, returns the id value, instead of the object. If there's no Field object with this name on the model, the model attribute's value is returned directly.

Used to serialize a field's value (in the serializer, or form output, for example). Normally, you would just access the attribute directly and not use this method.

`Talk.`**`speaker`**

`Talk.`**`tagged_items`**
This class provides the functionality that makes the related-object managers available as attributes on a model class, for fields that have multiple "remote" values and have a GenericRelation defined in their model (rather than having another model pointed *at* them). In the example "article.publications", the publications attribute is a ReverseGenericRelatedObjectsDescriptor instance.

`Talk.`**`tags = <taggit.managers._TaggableManager object at 0x7f5bf6dbe310>`**

`Talk.`**`unique_error_message`**(*model_class*, *unique_check*)

`Talk.`**`validate_unique`**(*exclude=None*)
Checks unique constraints on the model and raises `ValidationError` if any failed.

`admin` – Speakers admin

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

Formulario de media em linha

`can_delete` = **True**

`declared_fieldsets`

`exclude` = **None**

`extra` = **2**

`fields` = **None**

`fieldsets` = **None**

`filter_horizontal` = **()**

`filter_vertical` = **()**

`fk_name` = **None**

`form`
alias of `ModelForm`

`formfield_for_choice_field`(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a database Field that has declared choices.

`formfield_for_dbfield`(*db_field*, ***kwargs*)
Hook for specifying the form Field instance for a given database Field instance.

If kwargs are given, they're passed to the form Field's constructor.

`formfield_for_foreignkey`(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ForeignKey.

`formfield_for_manytomany`(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ManyToManyField.

`formfield_overrides` = **{}**

`formset`
alias of `BaseInlineFormSet`

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

get_formset(*request*, *obj=None*, ***kwargs*)
Returns a BaseInlineFormSet class for use in admin add/change views.

get_ordering(*request*)
Hook for specifying field ordering.

get_prepopulated_fields(*request*, *obj=None*)
Hook for specifying custom prepopulated fields.

get_readonly_fields(*request*, *obj=None*)
Hook for specifying custom readonly fields.

has_add_permission(*request*)

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

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

lookup_allowed(*lookup*, *value*)

max_num = None

media

model
alias of `MediaTalk`

ordering = None

prepopulated_fields = {}

queryset(*request*)

radio_fields = {}

raw_id_fields = ()

readonly_fields = ()

template = 'admin/edit_inline/tabular.html'

verbose_name = None

verbose_name_plural = None

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

Classe admin utilizada no django admin para oferecer as opcoes de CRUD do model Talk

action_checkbox(*obj*)
A list_display column containing a checkbox widget.

action_form
alias of `ActionForm`

actions = []

actions_on_bottom = False

actions_on_top = True

actions_selection_counter = True

add_form_template = None

add_view(**args*, ***kwargs*)
The 'add' admin view for this model.

change_form_template = None

change_list_template = None

change_view(**args*, ***kwargs*)
The 'change' admin view for this model.

changelist_view(**args*, ***kwargs*)
The 'change list' admin view for this model.

construct_change_message(*request*, *form*, *formsets*)
Construct a change message from a changed object.

date_hierarchy = 'created'

declared_fieldsets

delete_confirmation_template = None

delete_model(*request*, *obj*)
Given a model instance delete it from the database.

delete_selected_confirmation_template = None

delete_view(**args*, ***kwargs*)
The 'delete' admin view for this model.

exclude = None

fields = None

fieldsets = None

filter_horizontal = ()

filter_vertical = ()

form
alias of `ModelForm`

formfield_for_choice_field(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a database Field that has declared choices.

formfield_for_dbfield(*db_field*, ***kwargs*)
Hook for specifying the form Field instance for a given database Field instance.

If kwargs are given, they're passed to the form Field's constructor.

formfield_for_foreignkey(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ForeignKey.

formfield_for_manytomany(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ManyToManyField.

formfield_overrides = {}

get_action(*action*)
Return a given action from a parameter, which can either be a callable, or the name of a method on the ModelAdmin. Return is a tuple of (callable, name, description).

get_action_choices(*request, default_choices=[('', '———')]*)
Return a list of choices for use in a form object. Each choice is a tuple (name, description).

get_actions(*request*)
Return a dictionary mapping the names of all actions for this ModelAdmin to a tuple of (callable, name, description) for each action.

get_changelist(*request*, ***kwargs*)
Returns the ChangeList class for use on the changelist page.

get_changelist_form(*request*, ***kwargs*)
Returns a Form class for use in the Formset on the changelist page.

get_changelist_formset(*request*, ***kwargs*)
Returns a FormSet class for use on the changelist page if list_editable is used.

get_fieldsets(*request*, *obj=None*)
Hook for specifying fieldsets for the add form.

get_form(*request*, *obj=None*, ***kwargs*)
Returns a Form class for use in the admin add view. This is used by add_view and change_view.

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

get_inline_instances(*request*)

get_list_display(*request*)
Return a sequence containing the fields to be displayed on the changelist.

get_list_display_links(*request*, *list_display*)
Return a sequence containing the fields to be displayed as links on the changelist. The list_display parameter is the list of fields returned by get_list_display().

get_model_perms(*request*)
Returns a dict of all perms for this model. This dict has the keys `add`, `change`, and `delete` mapping to the True/False for each of those actions.

get_object(*request*, *object_id*)
Returns an instance matching the primary key provided. `None` is returned if no match is found (or the object_id failed validation against the primary key field).

get_ordering(*request*)
Hook for specifying field ordering.

get_paginator(*request*, *queryset*, *per_page*, *orphans=0*, *allow_empty_first_page=True*)

get_prepopulated_fields(*request*, *obj=None*)
Hook for specifying custom prepopulated fields.

get_readonly_fields(*request*, *obj=None*)
Hook for specifying custom readonly fields.

get_urls()

has_add_permission(*request*)
Returns True if the given request has permission to add an object. Can be overriden by the user in subclasses.

has_change_permission(*request*, *obj=None*)
Returns True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

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

has_delete_permission (*request*, *obj=None*)
Returns True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overriden 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.

history_view (*request*, *object_id*, *extra_context=None*)
The 'history' admin view for this model.

inlines = [<class 'talks.admin.MediaInline'>]

list_display = ('speaker', 'title', 'summary', 'created')

list_display_links = ()

list_editable = ()

list_filter = ('created',)

list_max_show_all = 200

list_per_page = 100

list_select_related = False

log_addition (*request*, *object*)
Log that an object has been successfully added.

The default implementation creates an admin LogEntry object.

log_change (*request*, *object*, *message*)
Log that an object has been successfully changed.

The default implementation creates an admin LogEntry object.

log_deletion (*request*, *object*, *object_repr*)
Log that an object will be deleted. Note that this method is called before the deletion.

The default implementation creates an admin LogEntry object.

lookup_allowed (*lookup*, *value*)

media

message_user (*request*, *message*)
Send a message to the user. The default implementation posts a message using the django.contrib.messages backend.

object_history_template = None

ordering = None

paginator
alias of `Paginator`

prepopulated_fields = {'slug': ('title',)}

queryset (*request*)
Returns a QuerySet of all model instances that can be edited by the admin site. This is used by changelist_view.

radio_fields = {}

raw_id_fields = ()

readonly_fields = ()

render_change_form(*request*, *context*, *add=False*, *change=False*, *form_url=''*, *obj=None*)

response_action(*request*, *queryset*)
Handle an admin action. This is called if a request is POSTed to the changelist; it returns an HttpResponse if the action was handled, and None otherwise.

response_add(*request*, *obj*, *post_url_continue='../%s/'*)
Determines the HttpResponse for the add_view stage.

response_change(*request*, *obj*)
Determines the HttpResponse for the change_view stage.

save_as = False

save_form(*request*, *form*, *change*)
Given a ModelForm return an unsaved instance. `change` is True if the object is being changed, and False if it's being added.

save_formset(*request*, *form*, *formset*, *change*)
Given an inline formset save it to the database.

save_model(*request*, *obj*, *form*, *change*)
Given a model instance save it to the database.

save_on_top = False

save_related(*request*, *form*, *formsets*, *change*)
Given the `HttpRequest`, the parent `ModelForm` instance, the list of inline formsets and a boolean value based on whether the parent is being added or changed, save the related objects to the database. Note that at this point save_form() and save_model() have already been called.

search_fields = ('title', 'created')

urls

2.7 Licença

Lorem ipsum

2.8 Contato

Lorem Ipsum

CHAPTER 3

Indices and tables

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

s

t