Email Client Documentation

Release 1.0

Hannon, Marcos Vinícius

January 27, 2016

Contents

Contents:

CHAPTER 1

push_email package

1.1 Subpackages

1.1.1 push_email.migrations package

Submodules

push_email.migrations.0001_initial module

class push_email.migrations.0001_initial.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = []

initial = True

operations = [<CreateModel fields=[('id', <django.db.models.fields.AutoField>), ('copy_to', <django.db.models.fields.E

push_email.migrations.0002_auto_20160111_2042 module

class push_email.migrations.0002_auto_20160111_2042.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('push_email', '0001_initial')]

operations = [<RenameField new_name='email_address', old_name='copy_to', model_name='copy'>, <RenameField

push_email.migrations.0003_auto_20160111_2043 module

class push_email.migrations.0003_auto_20160111_2043.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('push_email', '0002_auto_20160111_2042')]

operations = [<AlterField name='copies', model_name='myemail', field=<django.db.models.fields.related.ManyToMa

push_email.migrations.0004_emailid module

class push_email.migrations.0004_emailid.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('push_email', '0003_auto_20160111_2043')]

operations = [<CreateModel fields=[('id', <django.db.models.fields.AutoField>), ('email_id', <django.db.models.fields.

Module contents

1.2 Submodules

1.3 push_email.admin module

1.4 push_email.apps module

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

name = 'push_email'

1.5 push_email.calendarhandler module

1.6 push_email.forms module

class push_email.forms.**LoginForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*)
Bases: django.forms.forms.Form

base_fields = OrderedDict([('email_address1', <django.forms.fields.CharField object at 0x2afe6e6fc518>), ('passwor

declared_fields = OrderedDict([('email_address1', <django.forms.fields.CharField object at 0x2afe6e6fc518>), ('pas

media

1.7 push_email.models module

class push_email.models.**Copy**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Creates a "copy" table

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

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

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

In the example:

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

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

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

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

class push_email.models.**EmailId**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

A unique email ID

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

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

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

class push_email.models.**MyEmail**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Email object to save all useful email information

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

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

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

In the example:

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

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

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

MyEmail.**get_next_by_date**(**moreargs*, ***morekwargs*)

MyEmail.**get_previous_by_date**(**moreargs*, ***morekwargs*)

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

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

In the example:

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

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

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

class push_email.models.**Recipient**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Creates a "recipient" table

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

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

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

In the example:

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

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

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

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

1.8 push_email.tests module

class push_email.tests.**EmailTestCase**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

test_email_creation()

class push_email.tests.**ThreadTestCase**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

test_thread()

1.9 push_email.utils module

class push_email.utils.**EmailThread**(*username*, *password*, *imap_id*)
Bases: threading.Thread

check_for_new_emails()
Checks for new emails that match the pattern "[BBC423][xx.x.xxxx] Agenda dd/mm/aaaa hh:mm"

run()
Runs this thread

push_email.utils.**add_to_calendar**(*message*)
Formats an even to add it to google calendar :param message: an email message :return:

push_email.utils.**load_already_seen**()
Loads messages that have already been seen in previous runs of this code so that they don't appear as duplicate in the view for the user :return:

push_email.utils.**login**(*username*, *password*, *imap_id*)
Tries to log in and return the imbox object :param username: :param password: :param imap_id: :return: imbox

push_email.utils.**save_email**(*message*)
Saves a given email in the database :param message: email message :return:

1.10 push_email.views module

class push_email.views.**EmailLoginView**(**kwargs*)
Bases: django.views.generic.edit.FormView

Renders the login view

form_class
alias of LoginForm

success_url = '/home/'

template_name = 'index.html'

class push_email.views.**EmailView**(**kwargs*)
Bases: django.views.generic.base.View

Renders the email view page

post(*request*)
Gets a POST request with the login data and sends it to the backend :param request: POST request from the Login view :return: None

template_name = 'emails.html'

class push_email.views.**FullEmail**(*email_subject=None*, *email_from=None*, *email_body=None*, *email_date=None*, *recipients=None*, *copies=None*)
Bases: object

Creates a new Email object with all fields that will be used in the html templates

1.11 Module contents

CHAPTER 2

Models

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

p

P

R

S

T