django-pymess Documentation

Release 1.4

Luboš Mátl

Jul 13, 2018

Contents

Django-pymess is a library for sending various type of messages like: SMS, Push notifications or e-mails

CHAPTER 1

Project Home

https://github.com/druids/django-pymess

CHAPTER 2

Documentation

https://django-pymess.readthedocs.org/en/latest

2.1 Content

2.1.1 Installation

Using PIP

Django can use pip for installation:

```
$ pip install django-pymess
```

2.1.2 Configuration

After installation you must go through these steps:

Required Settings

The following variables have to be added to or edited in the project's `settings.py`:

For using pymess you just add `pymess` to `INSTALLED_APPS` variable:

```
INSTALLED_APPS = (
    ...
    'pymess',
    ...
)
```

Setup

SMS

PYMESS_SMS_TEMPLATE_MODEL
If you want to use your own SMS template model you must set this setting with your custom SMS template model that extends `pymess.models.sms.AbstractSMSTemplate` otherwise `pymess.models.sms.SMSTemplate` is used.

PYMESS_SMS_USE_ACCENT
Setting that sets if SMS will be sent with accent or not. Default value is `False`.

PYMESS_SMS_LOG_IDLE_MESSAGES
Setting that sets whether the delivery time is checked for messages. Default value is `True`.

SMS_SET_ERROR_TO_IDLE_MESSAGES
Setting that sets if idle messages will be moved to the error state after defined time. Default value is `True`.

PYMESS_SMS_IDLE_MESSAGES_TIMEOUT_MINUTES
If setting `PYMESS_SMS_LOG_IDLE_MESSAGES` is set to `True`, `PYMESS_SMS_IDLE_SENDING_MESSAGES_TIMEOUT_M` defines the number of minutes to send a warning that sms has not been sent. Default value is `10`.

PYMESS_SMS_DEFAULT_PHONE_CODE
Country code that is set to the recipient if phone number doesn't contain another one.

PYMESS_SMS_SENDER_BACKEND
Path to the SMS backend that will be used for sending SMS messages. Default value is `'pymess.backend.sms.dummy.DummySMSBackend'`.

PYMESS_SMS_ATS_CONFIG
Configuration of `pymess.backend.sms.ats_sms_operator.ATSSMSBackend`.

PYMESS_SMS_OPERATOR_CONFIG
Configuration of `pymess.backend.sms.sms_operator.SMSOperatorBackend`.

PYMESS_SMS_SNS_CONFIG
Configuration of `pymess.backend.sms.sns.SNSSMSBackend`.

E-MAIL

PYMESS_EMAIL_TEMPLATE_MODEL
If you want to use your own E-MAIL template model you must set this setting with your custom e-mail template model that extends `pymess.models.email.AbstractEmailTemplate` otherwise is used `pymess.models.email.EmailTemplate`.

PYMESS_EMAIL_SENDER_BACKEND
Path to the E-mail backend that will be used for sending e-mail messages. Default value is `'pymess.backend.emails.dummy.DummyEmailBackend'`.

PYMESS_EMAIL_BATCH_SENDING
If you use standard SMTP service you should send e-mails in batches otherwise other SMTP providers could add your SMTP server to the black-list. With this setting you configure e-mail backend not to send e-mails directly but messages are only created in state "waiting". Finally e-mails should be sent with Django command `send_emails_batch`. Default value is `False`.

PYMESS_EMAIL_BATCH_SIZE
Defines maximum number of e-mails that are sent with command `send_emails_batch`.

PYMESS_EMAIL_MANDRILL
Configuration of `pymess.backend.email.mandrill.MandrillEmailBackend`.

2.1.3 SMS

SMS messages that are stored inside Django model class defined later, are sent via SMS backend. There are implemented several SMS backends, every backed uses differend SMS service like twillio or AWS SNS. For sending SMS message you can use function `pymess.backend.sms.send` or `pymwess.backend.sms.send_template`.

`pymess.backend.sms.`**`send`**(*recipient*, *content*, *related_objects=None*, *tag=None*, ***sms_attrs*)
Function has two required parameters `recipient` which is a phone number of the receiver and `content`. Attribute `content` is a text message that will be sent inside the SMS body. If setting `PYMESS_SMS_USE_ACCENT` is set to `False`, accent in the content will be replaced by appropriate ascii characters. Attribute `related_objects` should contain a list of objects that you want to connect with the sent message (with generic relation). `tag` is string mark which is stored with the sent SMS message . The last non required parameter `**sms_kwargs` is extra data that will be stored inside SMS message model in field `extra_data`.

`pymess.backend.sms.`**`send_template`**(*recipient*, *slug*, *context_data*, *related_objects=None*, *tag=None*)
The second function is used for sending prepared templates that are stored inside template model (class that extends `pymess.models.sms.AbstractSMSTemplate`). The first parameter `recipient` is phone number of the receiver, `slug` is key of the template, `context_data` is a dictionary that contains context data for rendering SMS content from the template, `related_objects` should contains list of objects that you want to connect with the sent message and `tag` is string mark which is stored with the sent SMS message.

Models

class `pymess.models.sms.`**`OutputSMSMessage`**
The model contains data of already sent SMS messages.

`created_at`
Django `DateTimeField`, contains date and time of creation.

`changed_at`
Django `DateTimeField`, contains date and time the of last change.

`sent_at`
Django `DateTimeField`, contains date and time of sending the SMS message.

`recipient`
`CharField` that contains phone number of the receiver.

`sender`
`CharField` that contains phone number of the sender. Field can be empty if backend doesn't provide sender number.

`content`
`TextField`, contains content of the SMS message.

`template_slug`
If SMS was sent from the template, this attribute cointains key of the template.

`template`
If SMS was sent from the template, this attribute contains foreign key of the template. The reason why there is `template_slug` and `template` fields is that a template instance can be removed and it is good to keep at least the key of the template.

state
Field contains the current state of the message. Allowed states are:

- WAITING - SMS was not sent to the external service
- UNKNOWN - SMS was sent to the external service but its state is unknown
- SENDING - SMS was sent to the external service
- SENT - SMS was sent to the receiver
- ERROR - error was raised during sending of the SMS message
- DEBUG - SMS was not sent because system is in debug mode
- DELIVERED - SMS was delivered to the receiver

backend
Field contains path to the SMS backend that was used for sending of the SMS message.

error
If error was raised during sending of the SMS message this field contains text description of the error.

extra_data
Extra data stored with `JSONField`.

extra_sender_data
Extra data related to the SMS backend stored with `JSONField`. Every SMS backend can have different extra data.

tag
String tag that you can define during sending SMS message.

failed
Returns `True` if SMS ended in `ERROR` state.

related_objects
Returns DB manager of `pymess.models.sms.OutputSMSRelatedObject` model that are related to the concrete SMS message.

class pymess.models.sms.**OutputSMSRelatedObject**
Model for storing related objects that you can connect with the SMS message.

created_at
Django `DateTimeField`, contains date and time of creation.

changed_at
Django `DateTimeField`, contains date and time the of last change.

output_sms_message
Foreign key to the SMS message.

content_type
Content type of the stored model (generic relation)

object_id_int
If a related objects has primary key in integer format the key is stored here. This field uses db index therefore filtering is much faster.

object_id
Primary key of a related object stored in django `TextField`.

class pymess.models.sms.**AbstractSMSTemplate**
Abstract class of SMS template which you can use to define your own SMS template model. Your model that extends this class is set inside setting `PYMESS_SMS_TEMPLATE_MODEL`:

```
PYMESS_SMS_TEMPLATE_MODEL = 'your_application.YourSMSTemplateModel'
```

created_at
Django `DateTimeField`, contains date and time of creation.

changed_at
Django `DateTimeField`, contains date and time the of last change.

slug
Key of the SMS template in the string format (Django slug).

body
Body of the SMS message. Final SMS content is rendered with Django template system by default.

get_body()
Returns body of the model message. You can use it to update SMS body before rendering.

render_body(*context_data*)
Renders template stored inside `body` field to the message content. Standard Django template system is used by default.

can_send(*recipient*, *context_data*)
Returns by default `True` value. If you need to restrict sending SMS template for some reasons, you can override this method.

send(*recipient*, *context_data*, *related_objects=None*, *tag=None*)
Checks if message can be sent, renders message content and sends it via defined backend. Finally, the sent message is returned. If message cannot be sent, `None` is returned.

class pymess.models.sms.**SMSTemplate**
Default template model class that only inherits from `pymess.models.sms.AbstractSMSTemplate`

Backends

Backend is a class that is used for sending messages. Every backend must provide API defined by `pymess.backends.sms.SMSBackend` class. SMS backend is configured via `PYMESS_SMS_SENDER_BACKEND` (ex. `PYMESS_SMS_SENDER_BACKEND = 'pymess.backend.sms.sns.SNSSMSBackend'`). There are currently implemented following SMS backends:

class pymess.backend.sms.dummy.**DummySMSBackend**
Backend that can be used for testing. SMS is not sent, but is automatically set to the `DEBUG` state.

class pymess.backend.sms.sns.**SNSSMSBackend**
Backend that uses amazon SNS for sending messages (https://aws.amazon.com/sns/)

class pymess.backend.sms.twilio.**TwilioSMSBackend**
Backend that uses twilio service for sending SMS messages (https://www.twilio.com/)

class pymess.backend.sms.ats_sms_operator.**ATSSMSBackend**
Czech ATS SMS service is used for sending SMS messages. Service and backend supports checking if SMS was actually delivered. (https://www.atspraha.cz/)

Configuration of attributes according to ATS operator documentation:

```
PYMESS_ATS_SMS_CONFIG = {
    'URL': 'http://fik.atspraha.cz/gwfcgi/XMLServerWrapper.fcgi',  # If you use␣
↪default URL param, this doesn't need to be set
    'UNIQ_PREFIX': 'unique-id-prefix',  # If you use SMS service for more␣
↪applications you can define this prefix and it will be added to the message ID
```

(continues on next page)

(continued from previous page)

```
    'USERNAME': 'username',
    'PASSWORD': 'password',
    'UNIQ_PREFIX': '',
    'VALIDITY': 60,
    'TEXTID': None,
    'OPTID': '',
}
```

class pymess.backend.sms.sms_operator.**SMSOperatorBackend**

Czech SMS operator service is used for sending SMS messages. Service and backend supports checking if SMS was actually delivered. (https://www.sms-operator.cz/)

Configuration of attributes according to SMS operator documentation:

```
PYMESS_SMS_OPERATOR_CONFIG = {
    'URL': 'https://www.sms-operator.cz/webservices/webservice.aspx',  # If you␣
↪use default URL param, this doesn't need to be set
    'UNIQ_PREFIX': 'unique-id-prefix',  # If you uses SMS service for more␣
↪applications you can define this prefix and it will be added to the message ID
     'USERNAME': 'username',
     'PASSWORD': 'password',
}
```

Custom backend

If you want to write your own Pymess SMS backend, you must create class that inherits from `pymess.backends.sms.SMSBackend`:

```
.. class pymess.backends.sms.SMSBackend
```

publish_message(*message*)

This method should send SMS message (obtained from the input argument) and update its state. This method must be overridden in the custom backend.

publish_messages(*messages*)

If your service that provides sending messages in batch, you can override the `publish_messages` method. Input argument is a list of messages. By default, `publish_message` method is used for sending and messages are send one by one.

bulk_check_sms_states()

If your service provides checking SMS state you can override this method and implement code that check if SMS messages were delivered.

Commands

Because some services provide checking if SMS messages were delivered, Pymess provides a command that calls backend method `bulk_check_sms_state`. You can use this command inside cron and periodically call it. But SMS backend and service must provide it (must have implemented method `bulk_check_sms_states`).

2.1.4 E-mails

Like SMS E-mail messages are stored inside Django model class and sent via backend. Again we provide more e-mail backends, every backend uses different e-mail service like Mandrill, AWS SNS or standard SMTP. For send-

ing e-mail message you can use function `pymess.backend.email.send` or `pymwess.backend.email.send_template`.

`pymess.backend.emails.`**`send`**(*sender*, *recipient*, *subject*, *content*, *sender_name=None*, *related_objects=None*, *attachments=None*, *tag=None*, ***email_kwargs*)

Parameter `sender` define source e-mail address of the message, you can specify the name of the sender with optional parameter `sender_name`. `recipient` is destination e-mail address. Subject and HTML content of the e-mail message is defined with `subject` and `content` parameters. Attribute `related_objects` should contain a list of objects that you want to connect with the send message (with generic relation). Optional parameter `attachments` should contains list of files that will be sent with the e-mail in format `({file name}, {output stream with file content}, {content type})`. `tag` is string mark which is stored with the sent SMS message . The last non required parameter `**email_kwargs` is extra data that will be stored inside e-mail message model in field `extra_data`.

`pymess.backend.emails.`**`send_template`**(*recipient*, *slug*, *context_data*, *related_objects=None*, *attachments=None*, *tag=None*)

The second function is used for sending prepared templates that are stored inside template model (class that extends `pymess.models.sms.AbstractEmailTemplate`). The first parameter `recipient` is e-mail address of the receiver, `slug` is key of the template, `context_data` is a dictionary that contains context data for rendering e-mail content from the template, `related_objects` should contains list of objects that you want to connect with the send message, `attachments` should contains list of files that will be send with the e-mail and `tag` is string mark which is stored with the sent SMS message.

Models

class `pymess.models.emails.`**`EmailMessage`**

The model contains data of already sent e-mail messages.

`created_at`
Django `DateTimeField`, contains date and time of creation.

`changed_at`
Django `DateTimeField`, contains date and time the of last change.

`sent_at`
Django `DateTimeField`, contains date and time of sending the e-mail message.

`recipient`
`EmailField` that contains e-mail address of the receiver.

`sender`
`EmailField` that contains e-mail address of th sender.

`sender_name`
`CharField` that contains readable/friendly sender name.

`subject`
`TextField`, contains subject of the e-mail message.

`content`
`TextField`, contains content of the e-mail message.

`template_slug`
If e-mail was sent from the template, this attribute cointains key of the template.

`template`
If e-mail was sent from the template, this attribute contains foreign key of the template. The reason why there is `template_slug` and `template` fields is that a template instance can be removed and it is good to keep at least the key of the template.

state
Contains the current state of the message. Allowed states are:

- WAITING - e-mail was not sent to the external service
- SENDING - e-mail was sent to the external service
- SENT - e-mail was sent to the receiver
- ERROR - error was raised during sending of the e-mail message
- DEBUG - e-mail was not sent because system is in debug mode

backend
Field contains path to the e-mail backend that was used for sending of the SMS message.

error
If error was raised during sending of the SMS message this field contains text description of the error.

extra_data
Extra data stored with `JSONField`.

extra_sender_data
Extra data related to the e-mail backend stored with `JSONField`. Every SMS backend can have different extra data.

tag
String tag that you can define during sending SMS message.

failed
Returns `True` if SMS ended in `ERROR` state.

related_objects
Returns DB manager of `pymess.models.emails.EmailRelatedObject` model that are related to the concrete e-mail message.

class pymess.models.emails.**EmailRelatedObject**
Model for storing related objects that you can connect with the e-mail message.

created_at
Django `DateTimeField`, contains date and time of creation.

changed_at
Django `DateTimeField`, contains date and time the of last change.

email_message
Foreign key to the e-mail message.

content_type
Content type of the stored model (generic relation)

object_id_int
If a related objects have primary key in integer format the key is stored here. This field uses db index, therefore filtering is much faster.

object_id
Primary key of a related object stored in django `TextField`.

class pymess.models.emails.**Attachment**
Django model that contains e-mail attachments.

created_at
Django `DateTimeField`, contains date and time of creation.

changed_at
Django `DateTimeField`, contains date and time the of last change.

email_message
Foreign key to the e-mail message.

content_type
Django `CharField`, contains content type of the attachment.

file
Django `FileField`, contains file which was send to the recipient.

class pymess.models.emails.**AbstractEmailTemplate**
Abstract class of e-mail template which you can use to define your own e-mail template model. Your model that extends this class is set inside setting `PYMESS_EMAIL_TEMPLATE_MODEL`:

```
PYMESS_EMAIL_TEMPLATE_MODEL = 'your_application.YourEmailTemplateModel'
```

created_at
Django `DateTimeField`, contains date and time of creation.

changed_at
Django `DateTimeField`, contains date and time the of last change.

slug
Key of the e-mail template in the string format (Django slug).

sender
`EmailField` that contains e-mail address of the sender.

sender_name
`CharField` that contains readable/friendly sender name.

subject
`TextField`, contains subject of the e-mail message. Final e-mail subject is rendered with Django template system by default.

body
Body of the e-mail message. Final e-mail content is rendered with Django template system by default.

get_body()
Returns body of the model message. You can use it to update e-mail body before rendering.

render_body(*context_data*)
Renders template stored inside `body` field to the message content. Standard Django template system is used by default.

get_subject()
Returns subject of the model message. You can use it to update e-mail subject before rendering.

render_subject(*context_data*)
Renders template stored inside `subject` field to the message content. Standard Django template system is used by default.

can_send(*recipient*, *context_data*)
Returns by default `True` value. If you need to restrict sending e-mail template for some reasons, you can override this method.

send(*recipient*, *context_data*, *related_objects=None*, *tag=None*, *attachments=None*)
Checks if message can be sent, renders message content and sends it via defined backend. Finally, the sent message is returned. If message cannot be sent, `None` is returned.

class pymess.models.emails.**EmailTemplate**
Default template model class that only inherits from pymess.models.emails.AbstractEmailTemplate

Backends

Backend is a class that is used for sending messages. Every backend must provide API defined by pymess.backends.emails.EmailBackend class. E-mail backend is configured via PYMESS_EMAIL_SENDER_BACKEND (ex. PYMESS_EMAIL_SENDER_BACKEND = 'pymess.backend.emails.smtp.SMTPEmailBackend'). There are currently implemented following e-mail backends:

class pymess.backend.emails.dummy.**DummyEmailBackend**
Backend that can be used for testing. E-mail is not sent, but is automatically set to the DEBUG state.

class pymess.backend.emails.smtp.**SMTPEmailBackend**
Backend that uses standard SMTP service for sending e-mails. Configuration of SMTP is same as Django configuration.

class pymess.backend.emails.mandrill.**MandrillEmailBackend**
Backend that uses mandrill service for sending e-mail messages (https://mandrillapp.com/api/docs/index.python.html). For this purpose you must have installed mandrill library.

Configuration of attributes according to Mandrill operator documentation (the names of the configuration are the same):

```
PYMESS_EMAIL_MANDRILL_CONFIG = {
    'KEY': '',  # Mandrill notification key
    'HEADERS': None,
    'TRACK_OPENS': False,
    'TRACK_CLICKS': False,
    'AUTO_TEXT': False,
    'INLINE_CSS': False,
    'URL_STRIP_QS': False,
    'PRESERVE_RECIPIENTS': False,
    'VIEW_CONTENT_LINK': True,
    'ASYNC': False,
}
```

Custom backend

If you want to write your own Pymess e-mail backend, you must create class that inherits from pymess.backends.emails.EmailBackend:

```
.. class pymess.backends.sms.EmailBackend
```

publish_message(*message*)
This method should send e-mail message (obtained from the input argument) and update its state. This method must be overridden in the custom backend.

Commands

send_emails_batch

As mentioned e-mails can be sent in a batch with Django command send_emails_batch.

sync_emails

Store e-mail body in a HTML file is better from code readability. Therefore this command updates e-mails body from HTML files store in directory. You can select the directory with command property `directory` or you can set directory with setting `PYMESS_EMAIL_HTML_DATA_DIRECTORY`. E-mails body in the directory is stored like HTML file named with e-mail slug and html as a suffix.

dump_emails

E-mail body can be changed in the database therefore reverse operation to `sync_emails` can be done with this command. You must select directory where e-mails body in HTML format will be stored.

Index

G

O

P

R

S

T