django-db-mailer Documentation

Release 2.3.19

Aug 01, 2017

Contents

Django module to easily send emails using django templates stored on database.

CHAPTER 1

Contents

Installation

Compatibility

- Python: 2.7, pypy, 3.4, 3.5, 3.6, pypy3
- Django: 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11

Installation

Recommended way to install is via pip:

```
$ pip install django-db-mailer

# do not forget collect your project static on production servers
$ python manage.py collectstatic
```

Settings configuration

Add `dbmail` and `django.contrib.sites` to `INSTALLED_APPS` in the settings.py:

```
INSTALLED_APPS = (
    ...
    'django.contrib.sites',
    'dbmail',
    ...
)
SITE_ID = 1
```

DB initialization

Create application tables on database:

```
$ python manage.py syncdb
```

If you're using South:

```
$ python manage.py migrate
```

Important: South 1.0 or greater is required to run this migrations.

Settings

Required Settings

For minimize requests to database, configure django caches:

```
$ pip install redis django-pylibmc
    # or
$ pip install redis django-redis
```

```
# settings.py
CACHES = {
    # Memcached
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.PyLibMCCache',
    },
    # or Redis
    "default": {
        'BACKEND': 'redis_cache.cache.RedisCache',
        'LOCATION': '127.0.0.1:6379:2',
    },
    # or Memory
    "default": {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
        'LOCATION': 'unique-snowflake'
    },
}
```

Note: App do not work without caches

Configure project default SMTP settings:

```
# settings.py
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_HOST_USER = 'noreply@gmail.com'
EMAIL_HOST_PASSWORD = 'your-password'
EMAIL_USE_TLS = True
DEFAULT_FROM_EMAIL = 'User <noreply@gmail.com>'
```

Also you can configure smtp options for dbmail on the admin interface. But maybe other apps, like `django-registration` is used default project settings.

Optional Settings

Install `redis-server`, and configure `django-celery` for use priorities and scheduler:

```
$ pip install redis django-celery
```

```
# settings.py

import djcelery
import sys

INSTALLED_APPS += ('djcelery',)

BROKER_URL = 'redis://127.0.0.1:6379/1'

CELERY_ACKS_LATE = True
CELERYD_PREFETCH_MULTIPLIER = 1

# use priority steps only for mail queue
if 'mail_messages' in sys.argv:
    BROKER_TRANSPORT_OPTIONS = {
        'priority_steps': list(range(10)),
    }

CELERY_TASK_SERIALIZER = 'pickle'
CELERY_DEFAULT_QUEUE = 'default' # use mail_messages, if workers is divided

djcelery.setup_loader()
```

```
$ python manage.py celeryd --loglevel=debug -Q default
$ python manage.py celeryd --loglevel=info -Q mail_messages -n mail_messages # divide␣
↪workers and queues on production
```

Note: Do not forget define on command line queue name.

`django-db-mailer` can work without any third-party apps, but if you want to use all available app features and send emails on the background with priorities and scheduler, you need configure some apps, which will be pretty for your project and your clients.

Templates Revision:

```
$ pip install django-reversion
```

```
# settings.py
INSTALLED_APPS += ('reversion',)
```

Find information about compatibility with your Django versions here.

Templates Compare Revision:

```
$ pip install django-reversion-compare diff-match-patch
```

```
# settings.py
INSTALLED_APPS += ('reversion', 'reversion_compare',)
```

`django-reversion-compare` is not compatible at this time with Django 1.4+, but you can override `django-reversion-compare` templates on your project templates, and app will be work with Django 1.4+.

Editor:

```
$ pip install django-tinymce
# OR
$ pip install django-ckeditor
```

```
# settings.py
INSTALLED_APPS += ('tinymce',)
TINYMCE_DEFAULT_CONFIG = {
    'plugins': "table,spellchecker,paste,searchreplace",
    'theme': "advanced",
    'cleanup_on_startup': True,
    'custom_undo_redo_levels': 10,
}
# urls.py
urlpatterns += patterns(
    '', url(r'^tinymce/', include('tinymce.urls')),
)
```

Premailer:

```
$ pip install premailer
```

That's all what you need. App for turns CSS blocks into style attributes. Very pretty for cross-clients html templates.

Theme:

```
$ pip install django-grappelli
```

`django-db-mailer` supported from box `django-grappelli` and `django-suit` skin. Information about compatibility available here.

Translation Support:

```
$ pip install django-modeltranslation
```

```
# settings.py
MODELTRANSLATION_DEFAULT_LANGUAGE = 'en'
MODELTRANSLATION_LANGUAGES = ('ru', 'en')
MODELTRANSLATION_TRANSLATION_FILES = (
    'dbmail.translation',
)
INSTALLED_APPS = ('modeltranslation',) + INSTALLED_APPS

# If you are using django-grappelli, add grappelli_modeltranslation to the settings
INSTALLED_APPS = (
    'grappelli',
    'grappelli_modeltranslation',
    'modeltranslation',
) + INSTALLED_APPS
```

```
$ ./manage.py collectstatic
```

Update dbmail fields:

```
$ ./manage.py sync_translation_fields --noinput
```

Tracking:

```
$ pip install httpagentparser django-ipware
```

Add url patterns into urls.py:

```
urlpatterns += patterns(
    '', url(r'^dbmail/', include('dbmail.urls')),
)
```

Enable tracking and logging on settings:

```
DB_MAILER_TRACK_ENABLE = True
DB_MAILER_ENABLE_LOGGING = True
```

For track information about user, or about mail is read, you must be enable logging, and enable tracking on settings. Tracking templates must be HTML, not TXT. Celery workers must be launched, if celery is enabled. Django `sites` framework must be configured properly and have a real domain name record. `LibGeoIP` and `MaxMind` database must be installed and properly configured. To debug, open raw message and you can see html which specified on `DB_MAILER_TRACK_HTML`.

Usage

To use `django-db-mailer` on the your project - Add and Configure mail templates on the admin page.

Mail API

SendMail API with all available options:

```
from dbmail import send_db_mail

send_db_mail(
    # slug - which defined on db template
    slug='welcome',

    # recipient can be list, or str separated with comma or simple string
    # 'user1@example.com' or 'user1@example.com, user2@example.com' or
    # ['user1@example.com', 'user2@example.com'] or string Mail group slug
    recipient='user1@example.com',

    # All *args params will be accessible on template context
    {
        'username': request.user.username,
        'full_name': request.user.get_full_name(),
        'signup_date': request.user.date_joined
    },

    # You can access to all model fields. For m2m and fk fields, you should use␣
↪module_name
    MyModel.objects.get(pk=1),

    ######################################################################
    ### Optional kwargs:
    ######################################################################
    from_email='from@example.com',
    cc=['cc@example.com'],
```

```
    bcc=['bcc_1@example.com', 'bcc_2@example.com'],

    # For store on mail logs
    user=User.objects.get(pk=1),

    # Current language code, which selected by user
    language='ru',

    # This options is documented on the Django docs
    attachments=[(filename, content, mimetype)],
    files=['hello.jpg', 'world.png'],
    headers={'Custom-Header':'Some value'},

    # For working with different queue, you can specify queue name on the settings,␣
↪or on option
    queue='default',

    # Time for retry failed send. Working with celery only
    retry_delay=300,

    # Max retries, when sending is failed
    max_retries=3,

    # You can disable retry function
    retry=True,

    # Hard limit on seconds
    time_limit=30,

    # Postpone send email for specified seconds
    send_after=60,

    # You can disable celery for debug messages, or when error is occurred,
    # and email can not be delivered by celery.
    # Or some part of your app run on instance, where celery is not used.
    use_celery=True,

    # ...
    # another named arguments, which supported by specified backend
    # ...
)
```

‘‘slug‘‘ and ‘‘recipient‘‘ is not a named arguments.

For track information about read - add dbmail into urls.

SMS API

```
from dbmail import send_db_sms


send_db_sms(
    # slug which defined on db template
    slug='welcome',

    # recipient can be list, or str separated with comma or simple string
    # '+79031234567' or '+79031234567, +79031234568, +79031234569' or
```

```
    # ['+79031234567', '+79031234568'] or string Mail group slug
    recipient='+79031234567',

    # All *args params will be accessible on template context
    {
        'username': request.user.username,
        'full_name': request.user.get_full_name(),
        'signup_date': request.user.date_joined
    },

    # You can access to all model fields. For m2m and fk fields, you should use␣
↪module_name
    MyModel.objects.get(pk=1),

    # Optional kwargs:
    # from_email='DBMail'
    # user=User.objects.get(pk=1),
    #
    # language='ru',
    #
    # queue='default',
    # retry_delay=300,
    # max_retries=3,
    # retry=True,
    # time_limit=30,
    # send_after=60,
    #
    # use_celery=True,
)
```

TTS API

```
from dbmail import send_db_tts


send_db_tts(
    # slug which defined on db template
    slug='welcome',

    # recipient can be list, or str separated with comma or simple string
    # '+79031234567' or '+79031234567, +79031234568, +79031234569' or
    # ['+79031234567', '+79031234568'] or string Mail group slug
    recipient='+79031234567',

    # All *args params will be accessible on template context
    {
        'username': request.user.username,
        'full_name': request.user.get_full_name(),
        'signup_date': request.user.date_joined
    },

    # You can access to all model fields. For m2m and fk fields, you should use␣
↪module_name
    MyModel.objects.get(pk=1),

    # Optional kwargs:
```

```
    # from_email='DBMail'
    # user=User.objects.get(pk=1),
    #
    # language='ru',
    #
    # queue='default',
    # retry_delay=300,
    # max_retries=3,
    # retry=True,
    # time_limit=30,
    # send_after=60,
    #
    # use_celery=True,
)
```

Text to speech supported by default provider. But maybe not supported by your provider.

PUSH API

```
from dbmail import send_db_push


send_db_push(
    # slug which defined on db template
    slug='welcome',

    # recipient can be list, or str separated with comma or simple string
    # '+34cc3e5f0d2abf2ca0f9af170bd8cd2372a22f8a' or
↪'34cc3e5f0d2abf2ca0f9af170bd8cd2372a22f8a, 34cc3e5f0d2abf2ca0f9af170bd8cd2372a22f8b
↪' or
    # ['34cc3e5f0d2abf2ca0f9af170bd8cd2372a22f8a',
↪'34cc3e5f0d2abf2ca0f9af170bd8cd2372a22f8b'] or string Mail group slug
    recipient='34cc3e5f0d2abf2ca0f9af170bd8cd2372a22f8c',

    # All *args params will be accessible on template context
    {
        'username': request.user.username,
        'full_name': request.user.get_full_name(),
        'signup_date': request.user.date_joined
    },

    # You can access to all model fields. For m2m and fk fields, you should use␣
↪module_name
    MyModel.objects.get(pk=1),

    # Optional kwargs:
    # backend='dbmail.backends.push',
    # event='Server is down!',
    # from_email='ConsoleApp'
    # user=User.objects.get(pk=1),
    #
    # language='ru',
    #
    # queue='default',
    # retry_delay=300,
    # max_retries=3,
```

```
    # retry=True,
    # time_limit=30,
    # send_after=60,
    #
    # use_celery=True,
)
```

DBMail Backends

By default `django-dbmail` used 4 built-in backends (Mail/Sms/Tts/Push). But nothing prevents to write your own backend to work with all that you want.

Web API

You can use this app with different languages. For example on mobile apps, bash or some part of another languages without smtp access for notification & etc.

At first create API key for your app (you can do it from browser):

```
from dbmail.models import ApiKey

ApiKey.objects.create(name='Test', api_key='ZzriUzE')
```

Add urls route:

```
# urls.py
urlpatterns += patterns(
    '', url(r'^dbmail/', include('dbmail.urls')),
)
```

And send email from bash using `curl`:

```
$ apt-get install curl || brew install curl
$ curl -X POST http://127.0.0.1:8000/dbmail/api/ --data 'api_key=ZzriUzE&slug=welcome&
↪recipient=root@local.host'
```

or sms:

```
$ curl -X POST http://127.0.0.1:8000/dbmail/api/ --data 'api_key=ZzriUzE&slug=welcome&
↪recipient=%2B79031234567&backend=sms'
```

API bandwidth is 1k+ rps on i7 2.3GHz

DB template

Simple example to create template from shell:

```
from dbmail.models import MailTemplate

# Create new dbmail template.
MailTemplate.objects.create(
    name="Site welcome template",
    subject="[{{prefix}}] Welcome {{full_name}}",
    message="Hi, {{username}}. Welcome to our site.",
```

```
    slug="welcome",
    is_html=True,
)
```

Subscription API

Full stack (multiple) notification example for django.contrib.auth.models.users

```
from dbmail.models import MailSubscription
from dbmail import send_db_subscription

# Email notification
MailSubscription.objects.create(
    user_id=1,  # you can omit user_id if user not registered
    backend="dbmail.backends.mail",
    is_checked=True,
    address="user1@example.com"
)

# Push notification
MailSubscription.objects.create(
    user_id=1,
    backend="dbmail.backends.push",
    start_hour="08:00",
    end_hour="20:00",
    is_checked=True,
    defer_at_allowed_hours=True,
    address="d30NSrq10aO0hsyHDZ3"
)

# Send notification to all devices
send_db_subscription('welcome', 1)
```

If you want send notification for all subscribers, you can omit user_id

```
from dbmail.models import MailSubscription
from dbmail import send_db_subscription

# Subscribe nonexistent user for email notification
MailSubscription.objects.create(
    is_checked=True,
    address="user2@example.com"
)

# Subscribe nonexistent user for push notification
MailSubscription.objects.create(
    backend="dbmail.backends.push",
    is_checked=True,
    address="d30NSrq10aO0hsyHDZ4"
)

# Send notification to all available users (all devices)
send_db_subscription('welcome')
```

Send notification for all active users which registered at last 3 days ago (all devices):

```
from datetime import datetime, timedelta
from dbmail import send_db_subscription

send_db_subscription('welcome', None, {
    'user__is_active': 1,
    'user__date_joined__gte': datetime.now() - timedelta(days=3)
})
```

Send confirmation email message:

```
from dbmail.models import MailSubscription
from django.core.signing import dumps

# subscribe user
sub_obj = MailSubscription.objects.create(
    is_checked=False,
    address="user3@example.com"
)

# create hash code for confirmation
kwargs['hash_code'] = dumps({'pk': sub_obj.pk})

# send message (create MailTemplate)
MailSubscription.send_confirmation_link(
    slug='subs-confirmation', **kwargs
)
```

Create your own view for confirmation:

```
from dbmail.models import MailSubscription
from django.core.signing import loads

def confirmation(hash_code):
    data = loads(hash_code)
    sub_obj = MailSubscription.objects.get(pk=data['pk'])
    sub_obj.is_checked = True
    sub_obj.save()
```

Signals

Signals on database is a native Django signals.

Available variables for rules on Signals:

```
{{ date }} - current date
{{ date_time }} - current datetime
{{ site }} - current site
{{ domain }} - current site domain
{{ old_instance }} - old instance for pre_save
{{ current_instance }} - available when Update model state is enabled
{{ instance }} - instance from received signal
{{ ... }} - all instance fields as vars
```

When all signals was configured, you need to reload your wsgi application. Auto-reloading can be configured on settings by WSGI_AUTO_RELOAD/UWSGI_AUTO_RELOAD. But if you launch application on several instances, do it manually.

Note: Don't use a big intervals, if deferred tasks on queue more than 3-4k. It can crash a celery worker.

For deferred tasks best way is a crontab command + database queue. You can add `DB_MAILER_SIGNAL_DEFERRED_DISPATCHER = 'database'` into project settings, and call crontab command `send_dbmail_deferred_signal`.

Sending signals

```
from dbmail.exceptions import StopSendingException
from dbmail.signals import pre_send, post_send
from dbmail.backends.sms import Sender as SmsSender


def check_balance(*args, **kwargs):
    # "if" condition is unnecessary due to the way we connect signal to handler
    if kwargs['instance']._backend.endswith('sms'):
        balance = ...
        if balance == 0:
            raise StopSendingException

def decrease_balance(*args, **kwargs):
    # decrease user balance
    pass


pre_send.connect(check_balance, sender=SmsSender)
post_send.connect(decrease_balance, sender=SmsSender)
```

When you want transmit some **kwargs to signal, you can use *signals_kwargs*.

```
from dbmail import send_db_mail

send_db_mail(
    'welcome', 'user@example.com',
    signals_kwargs={'user': User.objects.get(pk=1)}, use_celery=False
)
```

When using MailSubscriptionAbstract model, you may want to check instance for uniqueness before creating it. In this case you should use either model meta option "unique-together" or use pre-save signal, that raises IntegrityError in case of duplicates.

```
from django.db import models

def check_address_is_unique(sender, instance, **kwargs):
    if not (instance.is_checked and instance.is_enabled):
        return

    query = sender.objects.filter(
        is_checked=True,
        is_enabled=True,
        address=instance.address,
        backend=instance.backend
    )
    if instance.pk:
        query = query.exclude(pk=instance.pk)
    if query.exists():
        raise IntegrityError('address must be unique')
```

```
models.signals.pre_save.connect(
    check_address_is_unique, sender=MailSubscription)
```

App settings

`django-db-mailer` has some configuration which can be configured on project:

```
# Default priority steps for admin.
# For more information about configure celery redis ``quasi-priorities``
# you can find here http://celery.readthedocs.org/en/latest/whatsnew-3.0.html#redis-
↪priority-support
DB_MAILER_PRIORITY_STEPS = (
    (0, _("High")),
    (3, _("Medium")),
    (6, _("Low")),
    (9, _("Deferred")),
)

# Best practice is use mail messages with priorities with different queue
# and worker. On this constants you can specify which queue will be using
# to send msg. For the different work you need specify workers on settings
# with tasks routing or etc. By default django-db-mailer used default
# queue. Do not forget run celery working with specified queue.
DB_MAILER_CELERY_QUEUE = 'default'
DB_MAILER_PUSH_QUEUE = 'default'
DB_MAILER_SMS_QUEUE = 'default'
DB_MAILER_TTS_QUEUE = 'default'
DB_MAILER_SUBSCRIPTION_QUEUE = 'default'

# By default celery is enabled. If djcelery is not on INSTALLED_APPS,
# this option is useless. When djcelery on INSTALLED_APPS, and you want
# disable it for some reason, you can change this constant.
# It very helpful for local development.
DB_MAILER_ENABLE_CELERY = True

# Show mail context into console. It very helpful for local development.
DB_MAILER_SHOW_CONTEXT = False

# Disable edit slug and notes on the admin. For production.
DB_MAILER_READ_ONLY_ENABLED = True

# Where attachments will be stored after uploading
DB_MAILER_UPLOAD_TO = 'mail_files'

# Default category on the admin
DB_MAILER_DEFAULT_CATEGORY = None

# Default email on the admin
DB_MAILER_DEFAULT_FROM_EMAIL = None

# Default priority for new templates
DB_MAILER_DEFAULT_PRIORITY = 6

# List per page on the admin template page
```

```
DB_MAILER_TEMPLATES_PER_PAGE = 20

# How much retries when error occurring by default
DB_MAILER_SEND_RETRY = 1

# Default delay before retry attempt
DB_MAILER_SEND_RETRY_DELAY = 300

# If celery is not used, this delay will be used to retry
DB_MAILER_SEND_RETRY_DELAY_DIRECT = 3

# Hard limit for send mail
DB_MAILER_SEND_MAX_TIME = 30

# Reload on the fly after Signal models will be changed
DB_MAILER_WSGI_AUTO_RELOAD = False
DB_MAILER_UWSGI_AUTO_RELOAD = False

# You can disable logging emails by default.
# This option override settings defined on the db templates
DB_MAILER_ENABLE_LOGGING = True

# Add app header. Very helpful for test app on production
DB_MAILER_ADD_HEADER = False

# Logs expire days for management command
DB_MAILER_LOGS_EXPIRE_DAYS = 7

# Models which will be show on the admin.
# Helpful when all features are not required
DB_MAILER_ALLOWED_MODELS_ON_ADMIN = [
    'MailFromEmailCredential',
    'MailFromEmail',
    'MailCategory',
    'MailTemplate',
    'MailLog',
    'MailGroup',
    'Signal',
    'ApiKey',
    'MailBcc',
]

# If you are using celery, and have a big mail queue,
# and admin can not be wait, when he receive test email,
# you can set False, and mail will be send without queue
DB_MAILER_USE_CELERY_FOR_ADMIN_TEST = True

# When inside invalidation not invalidate templates, you can use this
# constant, for automatically invalidation after defined seconds.
# By default cache invalidate only when admin update some templates.
DB_MAILER_CACHE_TIMEOUT = None

# We are strongly recommended use a different queue for signals, mail and mail on␣
↪signals
# Because on standard mail queue you will use priorities
# Big queues with countdown will constantly interfere and will be break, if priority␣
↪steps are to be used on current queue
DB_MAILER_SIGNALS_QUEUE = "default"
```

```
DB_MAILER_SIGNALS_MAIL_QUEUE = "default"

# For pending and very long task, you must use a database instead of the celery queues
DB_MAILER_SIGNAL_DEFERRED_DISPATCHER = 'celery'

# Remove database long tasks after execution
DB_MAILER_SIGNAL_DB_DEFERRED_PURGE = True

# Enable/Disable tracking functionality.
# If tracking is enabled, Logging must be enabled to.
# DbMail urls must be configured.
# Site framework must configured and installed.
DB_MAILER_TRACK_ENABLE = True

# Tracking image content and mime type
DB_MAILER_TRACK_PIXEL = [
    'image/gif',
    "\x47\x49\x46\x38\x39\x61\x01\x00\x01\x00\x80\x00"
    "\x00\xff\xff\xff\x00\x00\x00\x21\xf9\x04\x01\x00"
    "\x00\x00\x00\x2c\x00\x00\x00\x00\x01\x00\x01\x00"
    "\x00\x02\x02\x44\x01\x00\x3b"
]

# Html code for inject into message for tracking
DB_MAILER_TRACK_HTML = '<table bgcolor="white"><tr><td><font size="-1" color="black">
↪<img src="%(url)s" width="16" height="16" alt="" title="" border="0"></font></td></
↪tr></table></center>'

# Default backend for sending mail/sms/tts. You can redefine standard backend for␣
↪implement your custom logic.
DB_MAILER_BACKEND' = {
    'mail': 'dbmail.backends.mail',
    'tts': 'dbmail.backends.tts',
    'sms': 'dbmail.backends.sms',
}

# Default providers for sms and text to speech. If you want use different providers,␣
↪you can write simple function to do it. Look to examples at dbmail.providers.nexmo.
↪sms.
DB_MAILER_SMS_PROVIDER = 'dbmail.providers.nexmo.sms'
DB_MAILER_TTS_PROVIDER = 'dbmail.providers.nexmo.tts'
DB_MAILER_PUSH_PROVIDER = 'dbmail.providers.prowl.push'

# By default real api call is using.
# For log all requests to stdout - use True flag.
# Django DEBUG must be enabled.
DB_MAILER_DEBUG = False

# Default SMS from
DB_MAILER_DEFAULT_SMS_FROM = None

# Default Push notification from
DB_MAILER_DEFAULT_PUSH_FROM = None

# Apps which will be ignored on model browser
DB_MAILER_IGNORE_BROWSE_APP = [
    'south', 'dbmail', 'sessions', 'admin', 'djcelery',
    'auth', 'reversion', 'contenttypes'
```

```
]

# Function for transform html to text
DB_MAILER_MESSAGE_HTML2TEXT = 'dbmail.utils'

# Path to HTMLField class.
DB_MAILER_MODEL_HTMLFIELD = 'django.db.models.TextField'

# Path to MailSubscription class.
DB_MAILER_MAIL_SUBSCRIPTION_MODEL = 'dbmail.models.MailSubscription'

# You can use any backends designed as django email backend
# Example:
# - django.core.mail.backends.console.EmailBackend
# - postmark.django_backend.EmailBackend
# - django_ses.SESBackend and etc
# By default:
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'

# Subscription data field
DB_MAILER_MODEL_SUBSCRIPTION_DATA_FIELD = 'dbmail.fields.DataTextField'

# Default apns action
DB_MAILER_APNS_PROVIDER_DEFAULT_ACTION = 'Show'
```

Providers settings

Apple APNs

```
# Apple APNs provider settings
APNS_GW_HOST = 'gateway.sandbox.push.apple.com'  # or gateway.push.apple.com on␣
↪production
APNS_GW_PORT = 2195
APNS_CERT_FILE = 'cert.pem'                      # required. convert your p12 to pem
APNS_KEY_FILE = None

# Apple APNs via HTTP/2 protocol
APNS_GW_HOST = 'api.development.push.apple.com'  # or api.push.apple.com on production
APNS_GW_PORT = 443                               # or alternative 2197
APNS_CERT_FILE = 'cert.pem'                      # required. convert your p12 to pem
```

Google GCM

```
# Android GCM provider settings
GCM_KEY = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
```

Microsoft MPNs

```
# Windows MPNs provider settings
WP_CERT_FILE = None
```

Centrifugo

```
# Centrifugo provider settings
CENTRIFUGO_TOKEN = 'secret'
CENTRIFUGO_API = 'https://centrifugo.herokuapp.com/api/'
```

Nexmo

```
# nexmo.com (TTS and SMS)
NEXMO_USERNAME = ''
NEXMO_PASSWORD = ''
NEXMO_FROM = 'DBMail'
NEXMO_LANG = 'en-us'
```

Prowl

```
# prowlapp.com provider settings
PROWL_APP = 'DBMail'
```

Parse

```
# parse.com provider settings
PARSE_APP_ID = ""
PARSE_API_KEY = ""
```

PushOver

```
# pushover.net provider settings
PUSHOVER_TOKEN = ""
PUSHOVER_APP = "DBMail"
```

PubNub

```
# pubnub.com provider settings
PUBNUB_PUB_KEY = ""
PUBNUB_SUB_KEY = ""
PUBNUB_SEC_KEY = ""
```

Twilio

```
# twilio.com provider settings
TWILIO_ACCOUNT_SID = ""
TWILIO_AUTH_TOKEN = ""
TWILIO_FROM = ""
```

IQSms

```
# iqsms.ru provider settings
IQSMS_API_LOGIN = ""
IQSMS_API_PASSWORD = ""
IQSMS_FROM = ""
```

SmsAero

```
# smsaero.ru
SMSAERO_LOGIN = ""
SMSAERO_MD5_PASSWORD = ""
SMSAERO_FROM = ""
```

Slack/Mattermost

```
# slack.com / mattermost.org
SLACK_USERNAME = 'Robot'
SLACK_HOOCK_URL = 'https://hooks.slack.com/services/XXXXXXXXX/XXXXXXXXX/
↪XXXXXXXXXXXXXXXXXXXXXXXX'
SLACK_CHANNEL = 'main'
```

PushAll

```
# pushall.ru
PUSHALL_API_KEYS = {
    'default': {
        'title': 'AppName',
        'key': 'KEY',
        'id': 'ID',
        'priority': '1',
    }
}
```

SmsBliss

```
# smsbliss.ru/
SMSBLISS_API_URL = 'http://api.smsbliss.net/messages/v2/send.json'
SMSBLISS_LOGIN = ''
SMSBLISS_PASSWORD = ''
SMSBLISS_FROM = 'DbMail'
```

API

Objective-C example

```
NSURL *url = [NSURL URLWithString:@"http://127.0.0.1:8000/dbmail/api/"];
NSString *postString = @"api_key=ZzriUzE&slug=welcome&recipient=root@local.host";
NSData *returnData = [[NSData alloc]init];

NSMutableURLRequest *request = [[NSMutableURLRequest alloc] initWithURL:url];
[request setHTTPMethod:@"POST"];
[request setValue:[NSString stringWithFormat:@"%lu", (unsigned long)[postString␣
↪length]] forHTTPHeaderField:@"Content-length"];
[request setHTTPBody:[postString dataUsingEncoding:NSUTF8StringEncoding]];
returnData = [NSURLConnection sendSynchronousRequest: request returningResponse: nil␣
↪error: nil];

NSString *response = [[NSString alloc] initWithBytes:[returnData bytes]␣
↪length:[returnData length] encoding:NSUTF8StringEncoding];
NSLog(@"Response >>>> %@",response);
```

Java example

```
httpClient client = new DefaultHttpClient();
HttpPost post = new HttpPost("http://127.0.0.1:8000/dbmail/api/");

List<NameValuePair> pairs = new ArrayList<NameValuePair>();
pairs.add(new BasicNameValuePair("api_key", "ZzriUzE"));
pairs.add(new BasicNameValuePair("slug", "welcome"));
pairs.add(new BasicNameValuePair("recipient", "root@local.host"));
post.setEntity(new UrlEncodedFormEntity(pairs));

client.execute(post);
```

Python example

```
from httplib import HTTPConnection
from urlparse import urlparse
from urllib import urlencode

headers = {
    "Content-type": "application/x-www-form-urlencoded",
    "User-Agent": "DBMail Cli",
}

data = {
    "api_key": "ZzriUzE",
    "slug": "welcome",
    "recipient": "root@local.host"
}

uri = urlparse("http://127.0.0.1:8000/dbmail/api/")

http = HTTPConnection(uri.netloc)
```

```
http.request(
    "POST", uri.path,
    headers=headers,
    body=urlencode(data)
)
print http.getresponse().read()
```

Go example

```
package main

import (
    "net/http"
    "net/url"
    "bytes"
    "fmt"
)

func main() {
    uri := "http://127.0.0.1:8000/dbmail/api/"

    data := url.Values{}
    data.Add("api_key", "ZzriUzE")
    data.Add("slug", "welcome")
    data.Add("recipient", "root@local.host")

    client := &http.Client{}
    r, _ := http.NewRequest("POST", uri, bytes.NewBufferString(data.Encode()))
    r.Header.Set("Content-Type", "application/x-www-form-urlencoded")
    resp, _ := client.Do(r)
    fmt.Println(resp.Body)
}
```

PHP example

```
<?php
$url = 'http://127.0.0.1:8000/dbmail/api/';
$data = array(
    'api_key' => 'ZzriUzE', 'slug' => 'welcome', 'recipient' => 'root@local.host');
$options = array(
        'http' => array(
        'header'  => "Content-type: application/x-www-form-urlencoded\r\n",
        'method'  => 'POST',
        'content' => http_build_query($data),
    )
);

file_get_contents($url, false, stream_context_create($options));
```

using Curl

```
<?php
$url = 'http://127.0.0.1:8000/dbmail/api/';
$data = array(
```

```
    'api_key' => 'ZzriUzE', 'slug' => 'welcome', 'recipient' => 'root@local.host');

$ch = curl_init($url);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, http_build_query($data));
curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1);
curl_setopt($ch, CURLOPT_HEADER, 0);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

curl_exec($ch);
```

Ruby example

```
require "net/http"
require 'net/https'
require "uri"

uri = URI.parse("http://127.0.0.1:8000/dbmail/api/")
https = Net::HTTP.new(uri.host,uri.port)
req = Net::HTTP::Post.new(uri.path)

button = {
    "api_key" => "ZzriUzE",
    "slug" => "welcome",
    "recipient" => "root@local.host"
}
req.set_form_data(button)
https.request(req)
```

Node.js example

```
var request = require('request');

var uri = 'http://127.0.0.1:8000/dbmail/api/';
var data = {
    api_key: 'ZzriUzE',
    slug: 'welcome',
    recipient: 'root@local.host'
};

request.post({
    headers: {'content-type': 'application/x-www-form-urlencoded'},
    url: uri, form: data
}, function (error, response, body) {
    console.log(body);
});
```

Management commands

Commands

`send_dbmail_deferred_signal` - Send deferred mails which stored on database (if dbmail.Signals was used).

`update_dbmail_cache` - Best way for update cache after migration to new app version.

`clean_dbmail_cache` - Clear all caches.

`clean_dbmail_logs` - Clear old logs. Days can be defined as DB_MAILER_LOGS_EXPIRE_DAYS constant.

`dbmail_test_send` - Send test mail from command line. For example:

```
$ ./manage.py dbmail_test_send --email=root@local.host --pk=1 --without-celery
```

Crontab

Simple example:

```
SHELL=/bin/bash
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
MAILTO=root@localhost
PYTHON_BIN=/home/user/example.com/venv/bin/python
MANAGE_PY=/home/user/example.com/www/manage.py
LOG_FILE=/var/log/dbmail.cron.log

# Project commands
30 2 * * * $PYTHON_BIN $MANAGE_PY clean_dbmail_logs >> $LOG_FILE 2>&1
```

Browsers Web Push notifications

Supporting browsers

Desktop

- Chrome >= 50.0
- FireFox >= 44.0
- Safari >= 7

Mobile

- Google Chrome (Android only)

WebPush is working only with HTTPs.

Chrome/FireFox examples

Register your app on console.developers.google.com, and enable Cloud Messaging service. After registration you must be have `API KEY` and `APP ID`.

1. Add `GCM_KEY` with `API KEY` to the project settings:

```
GCM_KEY = 'AIza...'
```

2. Create `/static/manifest.json` with the following lines:

```
{
  "gcm_sender_id": "``{APP ID}``",
  "name": "Web Push Notification",
  "short_name": "WebPush",
  "icons": [
    {
      "src": "/static/images/icon-192x192.png",
      "sizes": "192x192"
    }
  ],
  "start_url": "/?homescreen=1",
  "display": "standalone",
  "gcm_user_visible_only": true,
  "permissions": [
    "gcm"
  ]
}
```

3. Add `/static/manifest.json` to meta on page

```
<head>
    <meta charset="UTF-8">
    ...
    <link rel="manifest" href="/static/manifest.json">
</head>
```

4. Create service worker `/static/js/service-worker.js` file

```
self.addEventListener('push', function (event) {
    if (event.data) {
        var payload = event.data.json();

        return self.registration.showNotification(payload.title, {
            body: data.body,
            icon: '/static/images/icon-192x192.png',
            data: payload,
        });
    }
});

self.addEventListener('notificationclick', function (event) {
    event.notification.close();

    if (event.notification.data && event.notification.data.url) {
        event.waitUntil(clients.matchAll({
            type: "window"
        }).then(function () {
            if (clients.openWindow) {
                return clients.openWindow(event.notification.data.url);
            }
        }));
    }
});
```

5. Register service worker to get permission and start background process

```
<head>
...
<script>
    function enableWebPush() {
        var is_chrome = navigator.userAgent.toLowerCase().indexOf('chrome') > -1;
        var is_ff = navigator.userAgent.toLowerCase().indexOf('firefox') > -1;
        if ((is_chrome || is_ff) && 'serviceWorker' in navigator) {
            navigator.serviceWorker.register('/service-worker.js').then(function () {
                navigator.serviceWorker.ready.then(function␣
↪(serviceWorkerRegistration) {
                    serviceWorkerRegistration.pushManager.getSubscription().
↪then(function (subscription) {
                        if (!subscription) {
                            serviceWorkerRegistration.pushManager.subscribe(
↪{userVisibleOnly: true}).then(function (subscription_info) {
                                var xhr = new XMLHttpRequest();
                                xhr.open("POST", "/dbmail/web-push/subscribe/", true);
                                xhr.setRequestHeader("Content-type", "application/x-
↪www-form-urlencoded");
                                xhr.setRequestHeader("X-CSRFToken", "{{ request.META.
↪CSRF_COOKIE }}");
                                xhr.send(JSON.stringify(subscription_info));

                                document.getElementById('subscription').innerHTML =␣
↪JSON.stringify(subscription_info);
                            });
                        }
                        document.getElementById('subscription').innerHTML = JSON.
↪stringify(subscription);
                    });

                });
            });
        }
    }
</script>
</head>
<body onload="enableWebPush()">
<div id="subscription"></div>
...
```

6. Open page to setup notification

```
$ open '/Applications/Google Chrome.app' --args https://localhost:8000/web-push/
```

7. Install `pywebpush` app

```
$ pip install 'pywebpush>=0.4.0'
```

8. Add Server-side Endpoints to `urls.py`

```
urlpatterns += patterns(
    '', url(r'^dbmail/', include('dbmail.urls')),
)
```

9. Add events receiver (subscribe/unsubscribe)

```
from dbmail import signals

def _web_push(**kwargs):
    # you must have your own function to store device token into db
    kwargs.pop('instance', None)
    kwargs.pop('sender', None)
    kwargs.pop('signal', None)
    print(kwargs)

signals.push_subscribe.connect(_web_push)
signals.push_unsubscribe.connect(_web_push)
```

10. And finally you can send notification from backend

```
from dbmail.providers.google.browser import send

subscription_info = {
    'endpoint': 'https://android.googleapis.com/gcm/send/dVj-T5fXaEw:AP...',
     'keys': {
        'auth': 'X6Ek_...',
        'p256dh': 'BBo..'
     }
}

send(subscription_info, message="Hello, World!", event="Python", url="..")
```

Safari examples

1. Register a Website Push ID (requires iOS developer license or Mac developer license)
2. Download and import to KeyChain the push notification certificate
3. Exporting Private Key .p12 from KeyChain
4. Generate .pem certificate to send notification via APNs

```
$ openssl pkcs12 -in apns-cert.p12 -out apns-cert.pem -nodes -clcerts
```

5. Check APNs connection

```
$ openssl s_client -connect gateway.push.apple.com:2195 -CAfile apns-cert.pem
```

6. Create contents of the Push Package

```
PushPackage.raw/
  icon.iconset
    icon_128x128@2x.png
    icon_128x128.png
    icon_32x32@2x.png
    icon_32x32.png
    icon_16x16@2x.png
    icon_16x16.png
  website.json
```

7. Contents of `website.json`

```
{
    "websiteName": "Localhost",
    "websitePushID": "web.ru.lpgenerator",
    "allowedDomains": ["https://localhost:8000"],
    "urlFormatString": "https://localhost:8000/%@",
    "authenticationToken": "19f8d7a6e9fb8a7f6d9330dabe",
    "webServiceURL": "https://localhost:8000/dbmail/safari"
}
```

8. Create Push Package (https://github.com/connorlacombe/Safari-Push-Notifications)

```
$ cp `php createPushPackage.php` pushPackages.zip
```

9. Add Server-side Endpoints to `urls.py`

```
urlpatterns += patterns(
    '', url(r'^dbmail/', include('dbmail.urls')),
)
```

10. Add events receiver (subscribe/unsubscribe/errors)

```
from dbmail import signals

def _safari_web_push(**kwargs):
    # you must have your own function to store device token into db
    kwargs.pop('instance', None)
    kwargs.pop('sender', None)
    kwargs.pop('signal', None)
    print(kwargs)

signals.safari_subscribe.connect(_safari_web_push)
signals.safari_unsubscribe.connect(_safari_web_push)
signals.safari_error_log.connect(_safari_web_push)
```

11. Add `APNS_GW_HOST` and `APNS_CERT_FILE` to the project settings

```
APNS_GW_HOST = 'api.push.apple.com'
APNS_GW_PORT = 443
APNS_CERT_FILE = 'apns-cert.pem'
APNS_KEY_FILE = None
```

12. Register service worker to get permission and start background process

```
<head>
...
<script>
    function enableSafariWebPush() {
        var websitePushID = "web.dev.localhost";
        var webServiceUrl = "https://localhost:8000/web-push/";
        var dataToIdentifyUser = {UserId: "123123"};

        var checkRemotePermission = function (permissionData) {
            if (permissionData.permission === 'default') {
                window.safari.pushNotification.requestPermission(
                        webServiceUrl,
                        websitePushID,
                        dataToIdentifyUser,
                        checkRemotePermission
```

```
                );
            }
            else if (permissionData.permission === 'denied') {
                console.dir(arguments);
                alert("Access denied. Please, enable push notification from Safari␣
↪settings.");
            }
            else if (permissionData.permission === 'granted') {
                document.getElementById('subscription').innerHTML = JSON.
↪stringify(permissionData.deviceToken);
            }
        };

        if ('safari' in window && 'pushNotification' in window.safari) {
            checkRemotePermission(
                window.safari.pushNotification.permission(websitePushID)
            );
        }
    }
</script>
</head>
<body onload="enableSafariWebPush()">
<div id="subscription"></div>
...
```

13. Open page to setup notification

```
$ open '/Applications/Safari.app' --args https://localhost:8000/web-push/
```

14. And finally you can send notification from backend

```
from dbmail import send_db_push

send_db_push(
    'welcome',
    '62B63D730C84E363627B95879CF13723B890249A4BA03BAC08004574DF17D2DA',
    use_celery=False,
    alert={
        "title": "Python",
        "body": "Hello, World!",
        "action": "View"
    }, **{"url-args": ["https://localhost:8000/admin/"]}
)
```

Local demo

You can test by demo which found on repo or use samples

1. Run server

```
$ cd demo
$ python manage.py runsslserver
```

2. Copy path to SSL certs

```
export CERT_PATH=`python -c 'import os, sslserver; print(os.path.dirname(sslserver.__
↪file__) + "/certs/development.crt")'`
echo $(CERT_PATH)
```

3. Import certs to KeyChain

Linux

```
sudo apt-get install -y ca-certificates
sudo cp "$CERT_PATH" /usr/local/share/ca-certificates/
sudo update-ca-certificates
```

OS X

```
sudo security add-trusted-cert -d -r trustRoot -k "/Library/Keychains/System.keychain
↪" "$CERT_PATH"
```

Windows (possible will not working)

```
certutil -addstore "Root" "$CERT_PATH"
    # or
certmgr.exe -add -all -c "$CERT_PATH" -s -r localMachine Root
```

4. Open demo url

```
$ open /Applications/Safari.app --args https://localhost:8000/web-push/
    # or
$ open '/Applications/Google Chrome.app' --args https://localhost:8000/web-push/
```

Custom backends and providers

Double backend example

A simple example of sending alerts by SMS and TTS.

```
from dbmail.backends.sms import Sender as SmsSender
from dbmail.backends.tts import Sender as TtsSender


class Sender(object):
    def __init__(self, *args, **kwargs):
        self.args = args
        self.kwargs = kwargs

    def send(self, is_celery=True):
        SmsSender(*self.args, **self.kwargs).send(is_celery)
        TtsSender(*self.args, **self.kwargs).send(is_celery)
        return 'OK'
```

```
class SenderDebug(Sender):
    def send(self, is_celery=True):
        print(self.args)
        print(self.kwargs)
        print({'is_celery': is_celery})


def send_double_notification(*args, **kwargs):
    from dbmail import db_sender

    kwargs['backend'] = 'demo.custom_backends.double'
    db_sender(*args, **kwargs)
```

Let's try:

```
from demo.custom_backends.double import send_double_notification

send_double_notification('welcome', '+79031234567')
```

Slack backend example

You're own backend, which send message to Slack channel.

```
from dbmail.backends.mail import Sender as SenderBase
from dbmail import import_module


class Sender(SenderBase):
    """
    Specify new backend when you want to change standard backends behavior
    More examples you can find at ./dbmail/backends directory
    """

    # you're custom provider will be defined here.
    # now we use standard provider
    provider = 'dbmail.providers.slack.push'

    # channels/recipients processing
    def _get_recipient_list(self, recipient):
        if isinstance(recipient, list):
            return recipient
        return map(lambda x: x.strip(), recipient.split(','))

    # send message
    def _send(self):
        module = import_module(self.provider)
        for recipient in self._recipient_list:
            module.send(recipient, self._message)


class SenderDebug(Sender):
    """
    Print message to stdout when DEBUG is True
    """
    def _send(self):
```

```
        self.debug('Message', self._message)


# helper function, which will be used on code
def send_db_slack(slug, *args, **kwargs):
    from dbmail import db_sender

    kwargs['backend'] = 'demo.custom_backends.slack'
    db_sender(slug, *args, **kwargs)
```

Slack settings

```
SLACK_USERNAME = 'robot'
SLACK_HOOCK_URL = 'https://hooks.slack.com/services/XXXXXXXXX/XXXXXXXXX/
↪XXXXXXXXXXXXXXXXXXXXXXXX'
SLACK_CHANNEL = 'main'
```

Let's try:

```
from demo.custom_backends.slack import send_db_slack

send_db_slack('welcome', {'username': 'GoTLiuM'})
```

Own provider

Create new file which will be implemented simple function below

```
def send(recipient, message, **kwargs):
    # some named args from send_db function
    custom_field = kwargs.pop('my_field', 'default value')
    ...
    # Some part of code, which will be send message over some protocol
    ...
    return True
```

Add necessary settings into settings.py

```
SOME_URL = '...'
```

Configure one of built-in backend to use your own provider

```
DB_MAILER_SMS_PROVIDER = 'apps.sms'
DB_MAILER_TTS_PROVIDER = 'apps.tts'
DB_MAILER_PUSH_PROVIDER = 'apps.push'
```

or write own function to use your provider

```
def send_over_own_provider(slug, *args, **kwargs):
    from dbmail import db_sender

    # can be one of built-in, or custom backend
    # kwargs['backend'] = 'dbmail.backends.sms'
    kwargs['provider'] = 'apps.sms'
    db_sender(slug, *args, **kwargs)
```

Providers examples

Apple APNs/APNs2

```
send_db_push(
    'welcome',
    'device-token',
    {
        'name': 'User'
    },
    provider='dbmail.providers.apple.apns',

    # ios specific
    category='NEW_MESSAGE_CATEGORY',
    content_available=0,
    sound='default',
    badge=6
)
```

Google GCM

```
send_db_push(
    'welcome',
    'user-id',
    {
        'name': 'User'
    },
    provider='dbmail.providers.google.android',

    # android specific
    vibrationPattern=[2000, 1000, 500, 500],
    ledColor=[0, 0, 73, 31],
    priority=2,
    msgcnt=2,
    notId=121,
)
```

Microsoft Tile

```
send_db_push(
    'welcome',
    'http://s.notify.live.net/u/1/sin/...',
    {
        'name': 'User'
    },
    provider='dbmail.providers.microsoft.tile',

    # MS specific
    template="tile",
    id="SecondaryTile.xaml?DefaultTitle=FromTile",
    count="5",
    back_background_image="http://background.com/back",
    background_image="http://background.images.com/background'",
```

```
    back_title="back title",
    back_content="back content here",
    event="title",  # instead title (configured on settings)
)
```

Microsoft Toast

```
send_db_push(
    'welcome',
    'http://s.notify.live.net/u/1/sin/...',
    {
        'name': 'User'
    },
    provider='dbmail.providers.microsoft.toast',

    # MS specific
    sound='',
    param='/Page2.xaml?NavigatedFrom=Toast Notification',
    path='/Views/MainScreen.xaml',
    event="title",  # instead title (configured on settings)
)
```

HTTP Push

```
send_db_push(
    'welcome',
    'http://localhost/receiver/',
    {
        'name': 'User'
    },
    provider='dbmail.providers.http.push',

    # Not limited args
    event='registration',
    uid='12345',
)
```

Centrifugo Push

```
send_db_push(
    'welcome',
    'users',
    {
        'name': 'User'
    },
    provider='dbmail.providers.centrifugo.push',

    # Not limited args
    event='registration',
    uid='12345',
)
```

PushAll Service

```
send_db_push(
    'welcome',
    'broadcast',
    {
        'name': 'User'
    },
    provider='dbmail.providers.pushall.push',

    # Not limited args
    title='MyApp',
    # uid='12345',  # only for unicast
    # icon='example.com/icon.png',
    # url='example.com',
    # hidden=0,
    # encode='utf8',
    # priority=1,
    # ttl=86400,
)
```

Demo installation

Docker

```
$ git clone --depth 1 -b master https://github.com/LPgenerator/django-db-mailer.git␣
↪db-mailer
$ cd db-mailer
$ docker build -t dbmail .
$ docker run -it -d -p 8000:8000 --name dbmail dbmail
$ docker exec -i -t dbmail /bin/bash
$ cd /mailer/
```

Vagrant

```
$ git clone --depth 1 -b master https://github.com/LPgenerator/django-db-mailer.git␣
↪db-mailer
$ cd db-mailer
$ vagrant up --provider virtualbox
$ vagrant ssh
$ cd /mailer/
```

OS X/Linux

```
$ sudo apt-get install -y virtualenvwrapper redis-server git python-dev libxml2-dev␣
↪libxslt-dev zlib1g-dev || brew install pyenv-virtualenvwrapper redis git
$ source /usr/share/virtualenvwrapper/virtualenvwrapper.sh || source /usr/local/bin/
↪virtualenvwrapper.sh
$ mkvirtualenv db-mailer
$ workon db-mailer
```

```
$ git clone --depth 1 https://github.com/LPgenerator/django-db-mailer.git db-mailer
$ cd db-mailer
$ python setup.py develop
$ cd demo
$ pip install -r requirements.txt
$ python manage.py syncdb --noinput
$ python manage.py migrate --noinput
$ python manage.py createsuperuser --username admin --email admin@local.host
$ redis-server >& /dev/null &
$ python manage.py runserver >& /dev/null &
$ python manage.py celeryd -Q default >& /dev/null &
```

Demo scenario

Open Shell:

```
$ python manage.py shell_plus --print-sql
```

Create new template:

```
from dbmail.models import MailTemplate
from dbmail import send_db_mail

MailTemplate.objects.create(
    name="Site welcome template",
    subject="Welcome",
    message="Welcome to our site. We are glad to see you.",
    slug="welcome",
    is_html=False,
)
```

Try to send test email with created template (without celery):

```
send_db_mail('welcome', 'user@example.com', use_celery=False)
```

Send email using celery:

```
send_db_mail('welcome', 'user@example.com')
```

Check mail logs:

```
from pprint import pprint
from django.forms.models import model_to_dict
from dbmail.models import MailLog

pprint([model_to_dict(obj) for obj in MailLog.objects.all()])
```

Open app in browser (login and password is admin/admin):

```
$ xdg-open http://127.0.0.1:8000/admin/dbmail/ >& /dev/null || open http://127.0.0.
↪1:8000/admin/dbmail/ >& /dev/null
```

Installation for development

Installation

Install all required packages and configure your project and environment:

```
$ sudo apt-get install -y virtualenvwrapper redis-server git python-dev libxml2-dev␣
↪libxslt-dev zlib1g-dev || brew install pyenv-virtualenvwrapper redis git
$ source /usr/share/virtualenvwrapper/virtualenvwrapper.sh || source /usr/local/bin/
↪virtualenvwrapper.sh
$ mkvirtualenv db-mailer
$ workon db-mailer
$ git clone --depth 1 https://github.com/LPgenerator/django-db-mailer.git db-mailer
$ cd db-mailer
$ python setup.py develop
$ cd demo
$ pip install -r requirements.txt
$ python manage.py syncdb --noinput
$ python manage.py migrate --noinput
$ python manage.py createsuperuser --username admin --email admin@local.host
$ redis-server >& /dev/null &
$ ln -sf /bin/bash /bin/sh
$ python manage.py runserver >& /dev/null &
$ python manage.py celeryd -Q default >& /dev/null &
$ python manage.py shell_plus --print-sql
```

Examples

Simple test from command line:

```
>>> from dbmail.models import MailTemplate, MailGroup, MailGroupEmail, MailLog
>>> from dbmail import send_db_mail

>>> MailTemplate.objects.create(
        name="Site welcome template",
        subject="Welcome",
        message="Welcome to our site. We are glad to see you.",
        slug="welcome",
        is_html=False,
    )

>>> group = MailGroup.objects.create(
        name="Site admins",
        slug="administrators",
    )
>>> MailGroupEmail.objects.bulk_create([
        MailGroupEmail(name="Admin 1", email="admin1@example.com", group=group),
        MailGroupEmail(name="Admin 2", email="admin2@example.com", group=group),
    ])

>>> # test simple string
>>> send_db_mail('welcome', 'root@localhost')

>>> # test emails list
>>> send_db_mail('welcome', ['user1@example.com', 'user2@example.com'])
```

```
>>> # test internal groups
>>> send_db_mail('welcome', 'administrators')

>>> # test without celery
>>> send_db_mail('welcome', 'administrators', use_celery=False)

>>> # Show what stored in logs
>>> print MailLog.objects.all().count()
```

Make targets

Simple shortcuts for fast development

`clean` - Clean temporary files
`clean-celery` - Clean all celery queues
`pep8` - Check code for pep8 rules
`sphinx` - Make app docs
`run` - Run Django development server
`run-celery` - Run celery daemon
`shell` - Run project shell
`run-redis` - Run Redis daemon
`help` - Display callable targets

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/LPgenerator/django-db-mailer/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

django-db-mailer could always use more documentation, whether as part of the official django-db-mailer docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/LPgenerator/django-db-mailer/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *django-db-mailer* for local development.

1. Fork the *django-db-mailer* repo on GitHub.
2. Clone your fork locally and switch to development branch:

```
$ git clone git@github.com:LPgenerator/django-db-mailer.git
$ cd django-db-mailer/
$ git fetch --all
$ git checkout development
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv django-db-mailer
$ workon django-db-mailer
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ make pep8
$ make test
$ make tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7 and for PyPy. Check https://travis-ci.org/LPgenerator/django-db-mailer/pull_requests and make sure that the tests pass for all supported Python versions.

Credits

Development Lead

- GoTLiuM InSPiRiT (gotlium)

Contributors

- Sergey Nikitin (nikitinsm)
- Volodymyr Tartynskyi (vosi)
- Petr Zelenin (petrikoz)
- icegreg (icegreg)
- Andrei (aloskutov-ki11)
- Sergey Khaylov (skhaylov)
- Ilya Streltsov (ilstreltsov)
- Antoine Nguyen (tonioo)
- Tomáš Peterka (katomaso)

CHAPTER 2

Contributing

You can grab latest version of code on `master` branch at Github.

Feel free to submit issues, pull requests are also welcome.

Good contributions follow simple guidelines