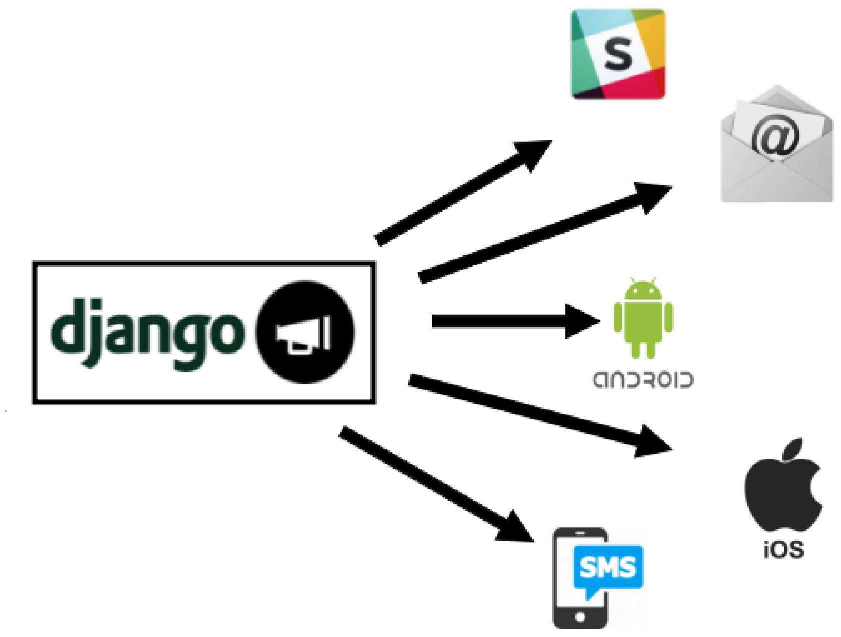
django-notifs

Release 2.6.4

Daniel Osaetin

Oct 12, 2021

CONTENTS



django-notifs is a modular notifications app for Django that basically allows you to notify users about events that occur in your application E.g

- Your profile has been verified
- User xxxx sent you a message

It also allows you to deliver these notifications to any destination you want to with custom delivery channels.

It also supports asynchronous notification with several pluggable delivery backends (e.g Celery, RQ etc)

CHAPTER

ONE

EXAMPLES?

A tutorial on how to build a Realtime Chat application with Vue, django-notifs, RabbitMQ and uWSGI

The Repository for the chat app (Chatire) is also available on github

CHAPTER

TWO

DOCUMENTATION

https://django-notifs.readthedocs.io

CHAPTER

THREE

CONTENTS

3.1 Overview

3.1.1 Requirements

- Python 3.6+
- Django 2.2+
- Celery
- Pika (RabbitMQ)

3.1.2 Supported Functionality

- In-app notifications
- Silent notifications (i.e Notifications that aren't saved in the database)
- Custom delivery (notification) channels e.g Telegram, Slack, SMS etc
- Asynchronous notifications (with support for multiple backends e.g Celery, RQ etc)

3.2 Installation

Get it from pip with:

```
pip install django-notifs
```

Include it in `settings.INSTALLED_APPS`:

```
INSTALLED_APPS = (
    'django.contrib.auth',
    ...
    'notifications',
    'django_jsonfield_backport'  # backport of JSONField for Django < 3.0
    ...
)
```

Finally don't forget to run the migrations with:

```
python manage.py migrate notifications
```

3.3 Usage

3.3.1 Quick start

To Create/Send a notification import the notify function and call it with the following arguments:

```python
from notifications.utils import notify

kwargs = {
    'source': self.request.user,
    'source_display_name': self.request.user.get_full_name(),
    'recipient': recipent_user, 'category': 'Chat',
    'action': 'Sent', 'obj': message.id,
    'short_description': 'You a new message', 'url': url,
    'channels': ('email', 'websocket', 'slack'), 'silent': True
}
notify(**kwargs)
```

This example creates a *silent* notification and delivers it via `email`, `websocket` and `slack`.

This assumes that you've implemented those channels and added them to the `settings.NOTIFICATIONS_CHANNELS` dictionary.

3.3.2 Notification Fields

The fields in the *args* dictionary map to the fields in the *Notification* model

- **source: A ForeignKey to Django's User model (Can be null if it's not a User to User Notification).**
- **source_display_name: A User Friendly name for the source of the notification.**
- **recipient: The Recipient of the notification. It's a ForeignKey to Django's User model.**
- **category: Arbitrary category that can be used to group messages.**
- **action: Verbal action for the notification E.g Sent, Cancelled, Bought e.t.c**
- **obj: The id of the object associated with the notification (Can be null).**
- **short_description: The body of the notification.**
- **url: The url of the object associated with the notification (Can be null).**
- **silent: If this Value is set, the notification won't be persisted to the database.**
- **extra_data: Arbitrary data as a dictionary.**
- **channels: Delivery channels that should be used to deliver the message (Tuple/List)**

The values of the fields can easily be used to construct the notification message.

3.3.3 Extra/Arbitrary Data

Besides the standard fields, django-notifs allows you to attach arbitrary data to a notification. Simply pass in a dictionary as the extra_data argument.

Note: The dictionary is serialized using python's json module so make sure the dictionary contains objects that can be serialized by the json module

Internally, the JSON is stored as plain text with django's standard `TextField`.

3.3.4 Writing custom delivery channels

django-notifs doesn't just allow you to send in-app notifications. you can also send external notifications (Like Emails and SMS) with custom delivery channels. A delivery channel is a python class that provides two methods:

1. `construct_message` to construct the message.
2. `notify` does the actual sending of the message.

There's a base meta class you can inherit. This is an example of an email delivery channel using *django.core.mail.send_mail*:

```python
from django.core.mail import send_mail
from notifications.channels import BaseNotificationChannel


class EmailNotificationChannel(BaseNotificationChannel):
    """Allows notifications to be sent via email to users."""

    def construct_message(self):
        """Constructs a message from notification arguments."""
        kwargs = self.notification_kwargs
        category = kwargs.get('category', None)
        short_description = kwargs.get('short_description', None)

        message = '{} {} {}'.format(
            kwargs['source_display_name'], kwargs['action'],
            short_description
        )

        return message

    def notify(self, message):
        """Send the notification."""
        subject = 'Notification'
        from_email = 'your@email.com'
        recipient_list = ['example@gmail.com']

        send_mail(subject, message, from_email, recipient_list)
```

Finally don't forget to tell *django-notifs* about your new Delivery Channel by setting:

```python
NOTIFICATIONS_CHANNELS = {
    'email': 'path.to.EmailNotificationChannel'
}
```

3.3.5 Sending notifications asynchronously

`django-notifs` is designed to support different backends for delivering notifications. By default it uses the `Synchronous` backend which delivers notifications synchronously.

Note: The Synchronous backend is not suitable for production because it blocks the request. It's more suitable for testing and debugging. To deliver notification asynchronously, please see the *backends section*.

3.3.6 Reading notifications

To read a notification use the read method:

```python
from notifications.utils import read

# id of the notification object, you can easily pass this through a URL
notify_id = request.GET.get('notify_id')

# Read notification
if notify_id:
    read(notify_id=notify_id, recipient=request.user)
```

Note: It's really important to pass the correct recipient to the `read` function.

Internally,it's used to check if the user has the right to read the notification. If you pass in the wrong recipient or you omit it entirely, `django-notifs` will raise a `NotificationError`

3.4 Configuration

3.4.1 `NOTIFICATIONS_CHANNELS`

`Default={}`

A dictionary of notification channels.

Keys: The softname of the notification channel which is used in your code

Values: The path to the notification channel's class.

Example:

```python
NOTIFICATIONS_CHANNELS = {
    'console': 'notifications.channels.ConsoleChannel'
}
```

django-notifs comes with an inbuilt console delivery channel that just prints out the notification arguments

3.4.2 `NOTIFICATIONS_DELIVERY_BACKEND`

`Default='notifications.backends.Synchronous`

`django-notifs` is designed to support different backends for delivering notifications. By default it uses the `Synchronous` backend which delivers notifications synchronously.

Note: The Synchronous backend is not suitable for production because it blocks the request. It's more suitable for testing and debugging. To deliver notification asynchronously, please see the *backends section*.

3.4.3 `NOTIFICATIONS_QUEUE_NAME`

`Default='django_notifs'`

This setting is only valid for the Celery, Channels and RQ backend

This is the queue name for backends that have a "queue" functionality

3.4.4 `NOTIFICATIONS_RETRY`

`Default=False`

Enable the retry functionality.

The Retry functionality is only valid for the Celery and RQ backends

`NOTIFICATIONS_RETRY_INTERVAL`

`Default=5`

The retry interval (in seconds) between each retry

`NOTIFICATIONS_MAX_RETRIES`

`Default=5`

The maximum number of retries for a notification.

3.4.5 `NOTIFICATIONS_WEBSOCKET_EVENT_NAME`

`Default='notifs_websocket_message'`

The `type` value of the messages that are going to received by the django notifs websocket consumer. In most cases, you don't need to change this setting.

3.4.6 NOTIFICATIONS_WEBSOCKET_URL_PARAM

`Default = 'room_name'`

The WebSocket URL param name. This setting **MUST** be used in the `notify` method's `extra_data` dictionary. It's also used to construct the WebSocket URL. See the *Advanced usage* section for more information.

3.5 Backends

The primary function of **a delivery backend** is to execute the code of the delivery channels. *Unlike delivery channels, you can only use one delivery backend at the same time.*

3.5.1 Celery

Install the optional Celery dependency with:

```
pip install django-notifs[celery]
```

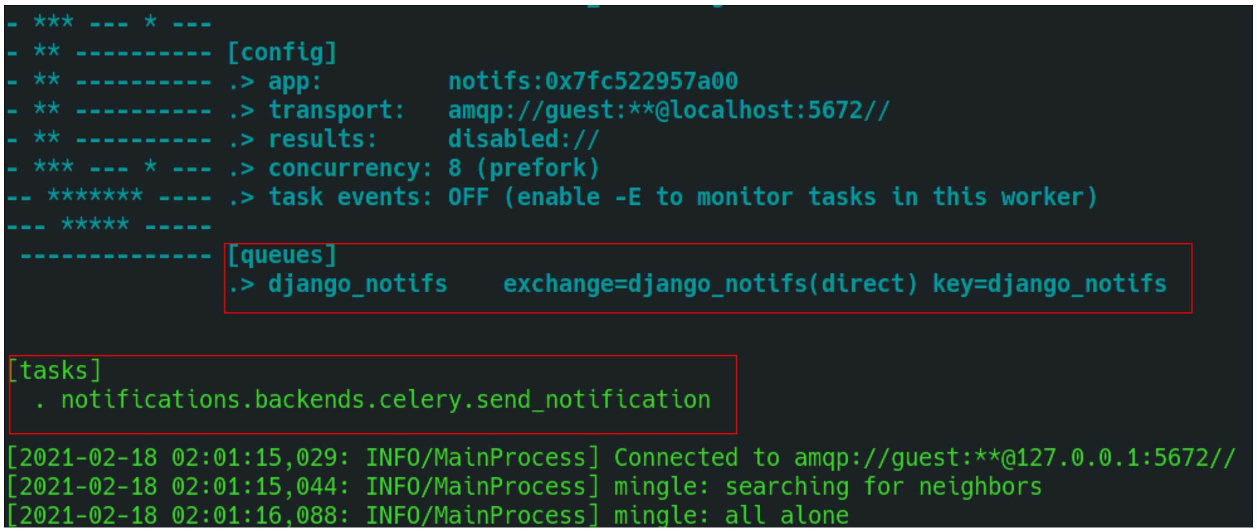
Enable it by setting `NOTIFICATIONS_DELIVERY_BACKEND` to `notifications.backends.Celery`

Run celery with the command:

```
celery -A yourapp worker -l info -Q django_notifs
```

Whenever a notification is created, it's automatically sent to celery and processed.

Make sure you see the queue and task (notifications.backends.celery.send_notification) in the terminal



```
- *** --- * ---
- ** ---------- [config]
- ** ---------- .> app:         notifs:0x7fc522957a00
- ** ---------- .> transport:   amqp://guest:**@localhost:5672//
- ** ---------- .> results:     disabled://
- *** --- * --- .> concurrency: 8 (prefork)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
 -------------- [queues]
                .> django_notifs    exchange=django_notifs(direct) key=django_notifs


[tasks]
  . notifications.backends.celery.send_notification

[2021-02-18 02:01:15,029: INFO/MainProcess] Connected to amqp://guest:**@127.0.0.1:5672//
[2021-02-18 02:01:15,044: INFO/MainProcess] mingle: searching for neighbors
[2021-02-18 02:01:16,088: INFO/MainProcess] mingle: all alone
```

If you have issues registering the task, you can import it manually or checkout the Celery settings in the repo.

3.5.2 Channels

Install the channels dependency with:

```
pip install django-notifs[channels]
```

This also installs channels_redis as an extra dependency

Declare the notifications consumer in `asgi.py`:

```
from notifications import consumers

application = ProtocolTypeRouter({
    ...,
    'channel': ChannelNameRouter({
        'django_notifs': consumers.DjangoNotifsConsumer.as_asgi(),
    })
})
```

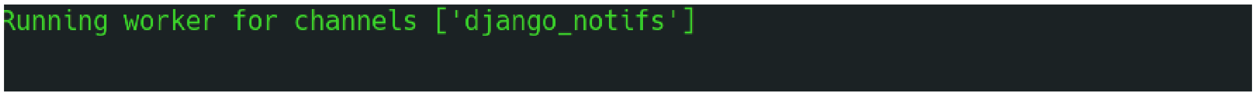
This example assumes that you're running Django 3x Which has native support for asgi. Check the channels documentation for Django 2.2

Next add the *django_notifs* channel layer to `settings.CHANNEL_LAYERS`:

```
CHANNEL_LAYERS = {
    ...,
    'django_notifs': {
        'BACKEND': 'channels_redis.core.RedisChannelLayer',
        'CONFIG': {
            "hosts": [('127.0.0.1', 6379)],
        },
    },
}
```

Finally, run the worker with:

```
python manage.py runworker django_notifs
```



```
Running worker for channels ['django_notifs']
```

3.5.3 RQ

RQ is a lightweight alternative to Celery. To use the RQ Backend, install the optional dependency with:

```
pip install django-notifs[rq]
```

django notifs uses django-rq under the hood

Enable it by setting `NOTIFICATIONS_DELIVERY_BACKEND` to `notifications.backends.RQ`

Configure the `django_notifs` in `settings.py`:

```
RQ_QUEUES = {
    ...,
    'django_notifs': {
        'HOST': 'localhost',
```

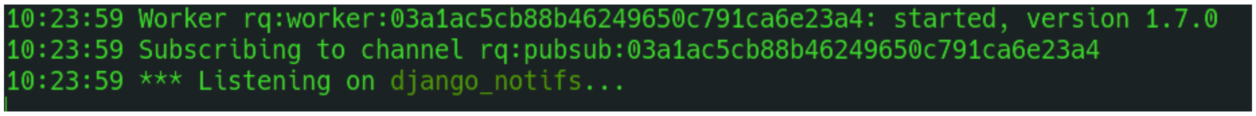
(continues on next page)

(continued from previous page)

```
        'PORT': 6379,
        'DB': 0,
        'PASSWORD': '',
        'DEFAULT_TIMEOUT': 360,
    }
}
```

Finally start the rq worker with:

```
python manage.py rqworker django_notifs --with-scheduler
```



```
10:23:59 Worker rq:worker:03a1ac5cb88b46249650c791ca6e23a4: started, version 1.7.0
10:23:59 Subscribing to channel rq:pubsub:03a1ac5cb88b46249650c791ca6e23a4
10:23:59 *** Listening on django_notifs...
```

See the django-rq documentation for more details

3.5.4 Synchronous

This is the default backend that sends notifications synchronously.

You can enable it explicitly by setting `NOTIFICATIONS_DELIVERY_BACKEND` to `notifications.backends.Synchronous`

3.6 Advanced usage

3.6.1 WebSockets

Unlike other django notification libraries that provide an API for accessing notifications, django-notifs supports websockets out of the box (thanks to *django-channels*). This makes it easy to send realtime notifications to your users in reaction to a new server side event.

If you're unfamiliar with django-channels. It's advised to go through the documentation so you can understand the basics.

3.6.2 Setting up the WebSocket server

This section assumes that you've already installed django-channels

Setup the consumer routing in your `asgi.py` file:

```
import os

import django
from django.core.asgi import get_asgi_application
from channels.routing import ProtocolTypeRouter, URLRouter

from notifications import routing as notifications_routing


os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'yourapp.settings')

```

(continues on next page)

(continued from previous page)

```
application = ProtocolTypeRouter({
    'http': get_asgi_application(),
    'websocket': URLRouter(notifications_routing.websocket_urlpatterns)
})
```

3.6.3 Notification channels

A simple WebSocket channel is provided:

- notifications.channels.WebSocketChannel

This channel simply delivers notifications to the `settings.NOTIFICATIONS_WEBSOCKET_EVENT_NAME` group.

Add the channel to `settings.NOTIFICATION_CHANNELS`:

```
NOTIFICATION_CHANNELS = {
    'websocket': 'notifications.channels.WebSocketChannel'
}
```

Sample usage:

```
from notifications import default_settings as notifs_settings
...

notif_args = {
    'source': user,
    'source_display_name': user.get_full_name(),
    'category': 'MESSAGE', 'action': 'Sent',
    'obj': obj.id,
    'short_description': 'You a new message', 'silent': True,
    'extra_data': {
        notifs_settings.NOTIFICATIONS_WEBSOCKET_URL_PARAM: chat_session.uri,
        'message': chat_session_message.to_json()
    }
}
notify(**notif_args, channels=['websocket'])
```

`notifs_settings.NOTIFICATIONS_WEBSOCKET_URL_PARAM` is a required key. You can also override it in `settings.py` and reference it through `django.conf.settings.NOTIFICATIONS_WEBSOCKET_URL_PARAM`

3.6.4 Running the WebSocket server

`ASGI` is capable of handling regular HTTP and WebSocket traffic so you don't really need to run a dedicated WebSocket server but it's still an option.

see the channels deployment documentation for more information on the best way to deploy your application.

3.6.5 How to listen to notifications

You listen to notifications by connecting to the WebSocket URL.

The default URL is `http://localhost:8000/<settings.NOTIFICATIONS_WEBSOCKET_URL_PARAM>`

To connect to a WebSocket room (via JavaScript) for a user `danidee` you'll need to connect to:

```
var websocket = new WebSocket('ws://localhost:8000/danidee')
```

You can always change the default route by Importing the `notifications.consumers.DjangoNotifsWebsocketConsumer` consumer and declaring another route. If you decide to do that, make sure you use the `NOTIFICATIONS_WEBSOCKET_URL_PARAM` setting because the Consumer class relies on it

an example to prefix the URL with `/chat` would be:

```
from django.urls import path

from . import default_settings as settings
from .consumers import DjangoNotifsWebsocketConsumer

websocket_urlpatterns = [
    path(
        f'chat/<{settings.NOTIFICATIONS_WEBSOCKET_URL_PARAM}>',
        DjangoNotifsWebsocketConsumer.as_asgi()
    )
]
```

3.6.6 Authentication?

This is out of the scope of django-notifs for now. This might change in the future as django-channels becomes more mature. Hence, The WebSocket endpoint is unprotected and you'll probably want to roll out your own custom authentication backend if you don't make use of the standard Authentication backend.

3.6.7 Testing and Debugging

django-notifs comes with an inbuilt console delivery channel that just prints out the notification arguments:

```
NOTIFICATIONS_CHANNELS = {
    'console': 'notifications.channels.ConsoleChannel'
}
```

This can be helpful during development where you don't want notifications to be delivered.