django-slackchat-serializer Documentation

Release 1.0.0

POLITICO

Feb 25, 2019

Contents:

Serialize conversations in Slack and notify your own custom renderer.

CHAPTER 1

Why this?

Slack is a great platform for liveblogging and creating custom threaded content. In the newsroom, we use it to cover live events and to break out features that take advantage of a more conversational tone.

There are two steps to publishing a slackchat. The first is to serialize the messages, reactions and threads you get from Slack into some data you can use to render a custom page. The second is to use that data to render the conversation on the page with whatever design conventions you choose and publish it wherever you need.

We separate those two concerns at POLITICO. Django-slackchat-serializer focuses on the former. It creates a rich, serialized representation of a conversation in Slack, including custom metadata you can add to individual messages.

Separating the serializer allows you to integrate it with your own custom renderer. To make the integration easy, slackchat-serializer throws off webhooks so your renderer can subscribe to updates and react accordingly, especially when publishing live.

Upshot, if you're looking to setup a slackchat system, this app removes the complexity of integrating with Slack so you can focus on the page design and business logic you need to publish your custom page or render within your CMS.

CHAPTER 2

Quickstart

2.1 Django configuration

Requires Django >= 2.0 and PostgreSQL >= 9.4.

1. Install django-slackchat-serializer using pip.

```
$ pip install django-slackchat-serializer
```

2. Add slackchat-serializer to your installed apps and create the minimum required token configuration to interact with Slack.

```
# project/settings.py

INSTALLED_APPS = [
    # ...
    'rest_framework',
    'foreignform',
    'slackchat',
]

SLACKCHAT_SLACK_VERIFICATION_TOKEN = os.getenv('SLACK_VERIFICATION_TOKEN')
SLACKCHAT_SLACK_API_TOKEN = os.getenv('SLACK_API_TOKEN')
```

3. Add slackchat to your project's `urls.py`.

```
# project/urls.py

urlpatterns = [
  # ...
  path('slackchat/', include('slackchat.urls')),
]
```

4. Slackchat uses Celery to process some tasks asynchronously. Read "First steps with Django" to see how to setup a Celery app in your project. Here is a configuration you can also use to start:

```
# project/celery.py
import os

from celery import Celery
from django.conf import settings

os.environ.setdefault('DJANGO_SETTINGS_MODULE', '<your project>.settings')

app = Celery('slackchat')
app.config_from_object('django.conf:settings', namespace='CELERY')
app.conf.update(
  task_serializer='json'
)
# Use synchronous tasks in local dev
if settings.DEBUG:
  app.conf.update(task_always_eager=True)
app.autodiscover_tasks(lambda: settings.INSTALLED_APPS, related_name='celery
↪')
```

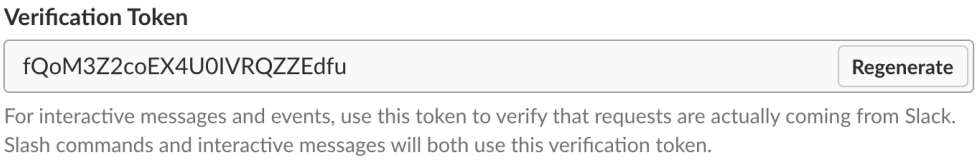
```
# project/__init__.py
from .celery import app as celery_app

__all__ = ['celery_app']
```

5. Run migrations.

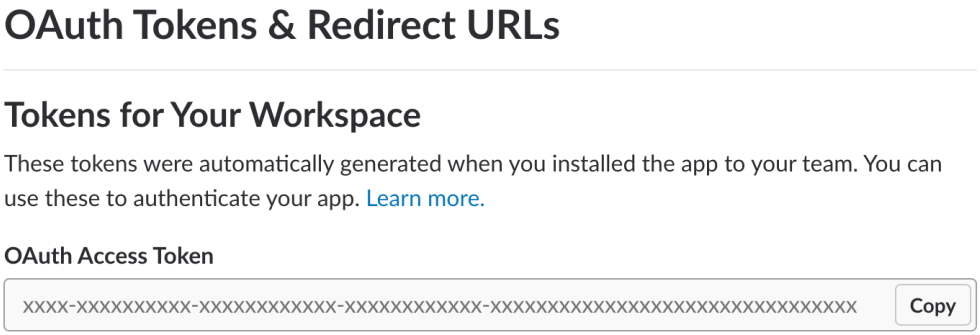
```
$ python manage.py migrate slackchat
```

2.2 Slack configuration

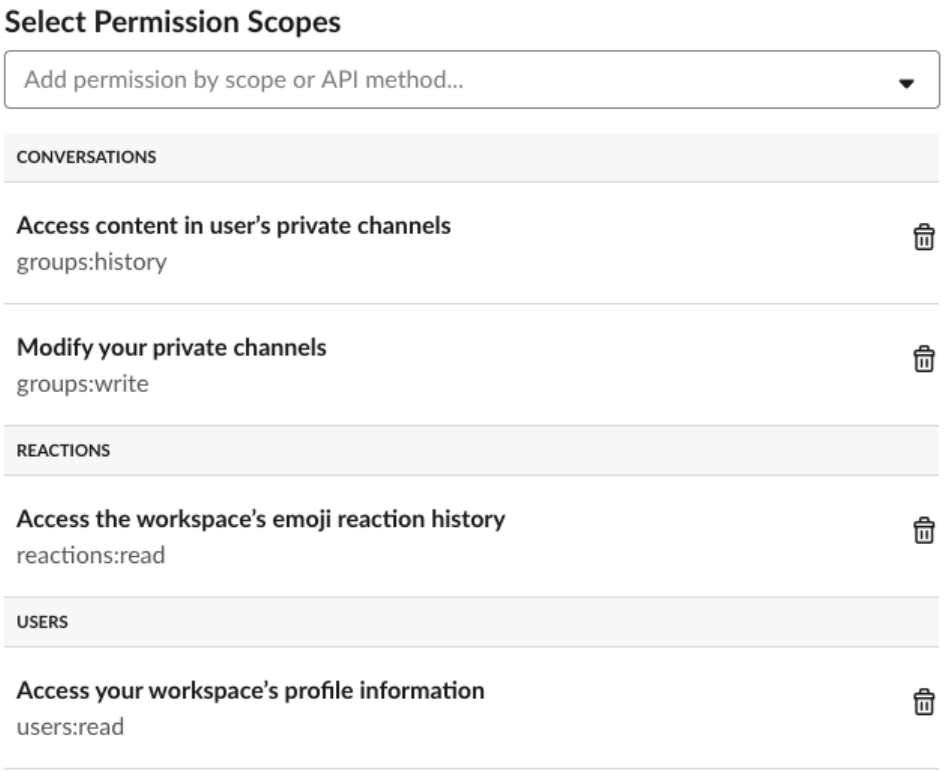
1. Create a new app for your team in Slack.
2. Grab your app's verification token (for `SLACKCHAT_SLACK_VERIFICATION_TOKEN`).



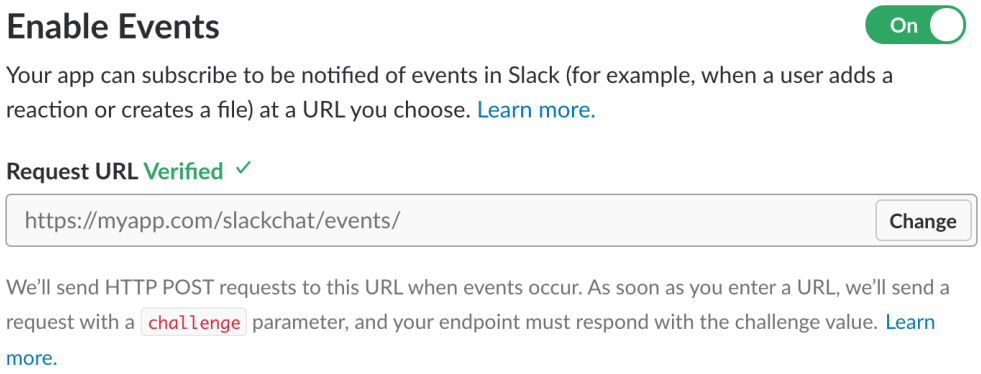
3. From the **OAuth & Permissions** section, get your app's OAuth Access Token (for `SLACKCHAT_SLACK_API_TOKEN`).



4. Add these permission scopes: `groups:history`, `groups:write`, `reactions:read`, `users:read`.



5. Enable events subscriptions in your app and configure the Request URL to hit slackchat-serializer's events endpoint. (Slackchat will automatically verify the URL with Slack.)



6. Subscribe to these workspace events: `message.groups`, `reaction_added` and `reaction_removed`.

Subscribe to Workspace Events

To subscribe to an event, your app must have access to the related OAuth permission scope.

Event Name	Description	Required Scope	
message.groups	A message was posted to a private channel	groups:history	
reaction_added	A member has added an emoji reaction to an item	reactions:read	
reaction_removed	A member removed an emoji reaction	reactions:read	

2.3 First slackchat

1. Run a management command to seed your app with Slack users:

   ` $ python manage.py get_slackchat_users `

2. Log into the Django admin.
3. Create a new `ChatType` instance.
4. Create a new `Channel` instance, which will create a new private channel in Slack to host your slackchat.
5. Invite any other members you want to the group and start chatting!

2.4 Configuring locally

If you're just trying out slackchat-serializer locally, we recommend using ngrok to proxy Django's own development server to a public URL that Slack can hit with event messages.

If you're running your development server on port 8000, for example, you can start an ngrok tunnel like this:

```
$ ngrok http 8000
```

Now grab the tunnel's `https` URL and use it to configure the request URL in your Slack app's event subscriptions.

2.5 Developing

Move into the example directory and start a pipenv shell.

```
$ pipenv shell
```

You can now develop using Django's development server.

CHAPTER 3

Configuration options

3.1 Required config

3.1.1 SLACKCHAT_SLACK_VERIFICATION_TOKEN

Slack app verification token.

3.1.2 SLACKCHAT_SLACK_API_TOKEN

Slack app OAuth access token.

3.1.3 SLACKCHAT_PUBLISH_ROOT

The URL root of your front end. It will be combined with the `publish_path` of both the `ChatType` and `Channel` to create preview links in the CMS.

For example, if you have a `ChatType` with the `publish_path` of `/my-custom-type/` and your channel's `publish_path` is `my-chat` you might expect that this chat will be published at `https://my-site.com/slack-chats/my-custom-type/my-chat/`. In this case your `SLACKCHAT_PUBLISH_ROOT` would be `https://my-site.com/slack-chats/`.

3.1.4 SLACKCHAT_TEAM_ROOT

The URL for your Slack team. It should take the form of `https://your-team.slack.com`. This is also used to create links in your CMS.

3.1.5 SLACKCHAT_DEFAULT_OWNER

The Slack user ID for the default owner of new Slack Chats. While the CMS requires that user's be logged in, if the logged-in Django user does not have a matching Slackchat user, this default owner will be set as the owner and invited to any new chats.

3.2 Additional config

3.2.1 SLACK_WEBHOOK_VERIFICATION_TOKEN

A custom token that will be sent in webhook notification post data as `token`. This can be used to verify requests to your renderer's endpoint come from slackchat.

```
# default
SLACK_WEBHOOK_VERIFICATION_TOKEN = 'slackchat'
```

3.2.2 SLACKCHAT_PUBLISH_ROOT

Configuring this to the URL root where your slackchats are published by a renderer will add a direct link to each chat in the `Channel` Django admin.

```
# e.g.
SLACKCHAT_PUBLISH_ROOT = 'https://mysite.com/slackchats/'
```

3.2.3 SLACK_MARKSLACK_USER_TEMPLATE

A function used to create a user_templates object in markslack. The function should take a `User` instance argument and return a formatted string.

```
# default
SLACK_MARKSLACK_USER_TEMPLATE = lambda user: '<span class="mention">{} {}</span>'.
↪format(
    user.first_name,
    user.last_name
)
```

3.2.4 SLACK_MARKSLACK_LINK_TEMPLATES

A markslack link_templates object.

```
# default
SLACK_MARKSLACK_LINK_TEMPLATES = {
      'twitter.com': '<blockquote class="twitter-tweet" data-lang="en"><a href="{}"></
↪a></blockquote>',
  }
```

3.2.5 SLACK_MARKSLACK_IMAGE_TEMPLATE

A markslack image_template object.

```
# default
SLACK_MARKSLACK_IMAGE_TEMPLATE = '<figure><img href="{}" /></figure>'
```

3.2.6 SLACKCHAT_USER_IMAGE_UPLOAD_TO

A function used to set the upload path used when copying Slack user profile images to your own server.

```
# default
def default_user_image_upload_to(instance, filename):
    return 'slackchat/users/{0}{1}/{2}'.format(
        instance.first_name,
        instance.last_name,
        filename,
    )

SLACKCHAT_USER_IMAGE_UPLOAD_TO = default_user_image_upload_to
```

3.2.7 SLACKCHAT_MANAGERS

An array of Slack User IDs. These users will be automatically invited to any new channels made by this app.

```
# default
SLACKCHAT_MANAGERS = []
```

3.2.8 SLACKCHAT_CMS_TOKEN

A token used to authenticate API requests made by the CMS. Defaults to a random hash created on server startup, but you can use this setting to set it explicitly if you want to use the API endpoint to create/update Channel models.

```
# default
SLACKCHAT_CMS_TOKEN = "%032x" % random.getrandbits(128)
```

CHAPTER 4

Models

4.1 ChatType

A type of slackchat.

4.2 Channel

A channel that hosts a slackchat.

4.3 User

A participant in a slackchat.

4.4 Message

A message posted by a user in a Slack channel.

4.5 Reaction

An emoji reaction to a message in Slack.

4.6 Attachment

An unfurled link or media item attached to a message.

4.7 Argument

An argument that can be attached to a message through an emoji reaction in Slack. Arguments can signal to a render that a message should be handled in a special way.

4.8 KeywordArgument

A keyword argument that can be attached to a message through a threaded message. Keyword arguments can signal to a render that a message should be handled in a special way.

4.9 CustomContentTemplate

A template that matches a regex pattern against a message's content and can be used to reformat that content or to attach an argument when a match is found.

CHAPTER 5

Serialization

5.1 Channel

Slackchat-serializer includes a RESTful API with a single endpoint used to serialize a channel and all its messages, reactions and users.

You can access the serialized representation for any channel at:

`{slackchat URL}/api/channels/{channel ID}/`

Here's an example of a serialized channel:

```
{
  "id": "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
  "api_id": "GXXXXXXXX",
  "chat_type": "basic",
  "title": "Our first Slackchat!",
  "introduction": "Welcome to our first Slackchat. \nFollow along below:",
  "meta": {
      "title": "First Slackchat",
      "description": "Live blogging.",
      "image": "http://mysite.com/share-image.jpg"
  },
  "extras": {
    "video": "https://www.youtube.com/watch?v=V7uEb_XrK1U"
  },
  "paths": {
      "channel": "/2018-01-01/some-chat/",
      "chat_type": "/slackchats/"
  },
  "publish_path": "/slackchats/2018-01-01/some-chat/",
  "publish_time": "2018-01-01T01:30:00Z",
  "live": true,
  "users": {
      "U4XV32XKR": {
```

(continues on next page)

(continued from previous page)


```
            "first_name": "Jon",
            "last_name": "McClure",
            "image": "slackchat/users/JonMcClure/profile-7f37ceefad.jpg",
            "title": "Interactive news editor"
        }
    },
    "messages": [
        {
            "timestamp": "2018-02-04T15:00:45.000065Z",
            "user": "U4XV32XKR",
            "content": "Hi, welcome to our **first** *Slackchat*!",
            "attachments": [
              {
                  "image_url": "https://myserver.com/some-image.jpg",
                  "image_width": 400,
                  "image_height": 400
              }
            ],
            "args": ["edited"],
            "kwargs": {
              "style": "moderator-styles"
            }
        },
        {
            "timestamp": "2018-02-04T15:10:09.000129Z",
            "user": "U4XV32XKR",
            "content": "Check out this [link](http://www.google.com).",
            "reactions": [
              {
                "timestamp": "2018-02-04T19:21:29.000085Z",
                "reaction": "fire",
                "user": "U4XV32XKR"
              },
            ],
        }
    ],
    "timestamp": "2018-01-01T23:46:26.321994Z"
}
```


5.1.1 chat_type

Slackchats are serialized with a chat_type property, which represents a `ChatType` instance.

This is useful to identify a particular template your renderer may use to render different types of slackchats, for example, using different design treatments or branding.

Custom *args & kwargs* patterns are also configured per `ChatType`.

5.1.2 meta

Use meta attributes to fill out social meta tags in your renderer.

5.1.3 extras

You can configure additional fields per ChatType to collect specialized data for a channel. Say you have a ChatType that includes a live video feed. You can configure a field to collect a video embed code for this template, and the value for that field will be serialized here.

Note: Configure additional fields in the `ChatType` admin using a JSON Schema and UI Schema. Users will then see the additional fields in the `Channel` admin. See django-foreignform for more information on using this feature.

5.1.4 messages

Messages are rendered in Markdown syntax by default, unless `render_to_html = True` on the `ChatType` instance, in which case the message is rendered from Markdown into HTML when serializing.

See the markslack package and *Configuration options* for more information on how your users can format links, images, user mentions and text in Slack messages.

5.1.5 reactions

Reactions are captured with the emoji code of the reaction, for example, `fire` for .

We recommend using the emoji package to translate reaction emoji codes to true unicode symbols in your renderer, which is what markslack uses when converting messages from Slack.

5.1.6 attachments

Attachments are links or images Slack has "unfurled" in a message. They contain metadata about a link or media item that allows you to render it in a richer way.

See Slack for more information.

5.1.7 args & kwargs

With each message you can serialize custom data, which can signal some special handling to your renderer.

Slackchat-serializer lets you construct that data like the arguments and keyword arguments you'd pass to a function. Configure them using the `Argument` and `KeywordArgument` models and then consume them in your renderer.

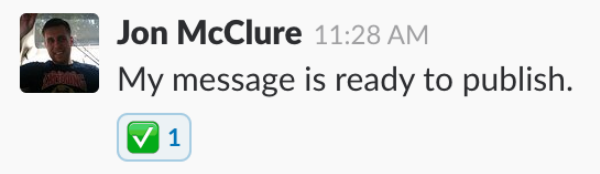
These features can be used to represent important workflow steps or to add custom metadata to messages.

args

Args are most often created through emoji reactions in Slack.

For example, say you want the `:white_check_mark:` ( ) reaction to signal to your renderer that a message has been copyedited.

You can create an `Argument` object associated with that character – e.g., `'white_check_mark'` – with a custom argument name – e.g., `'edited'` – that will be serialized with any message with that emoji reaction.



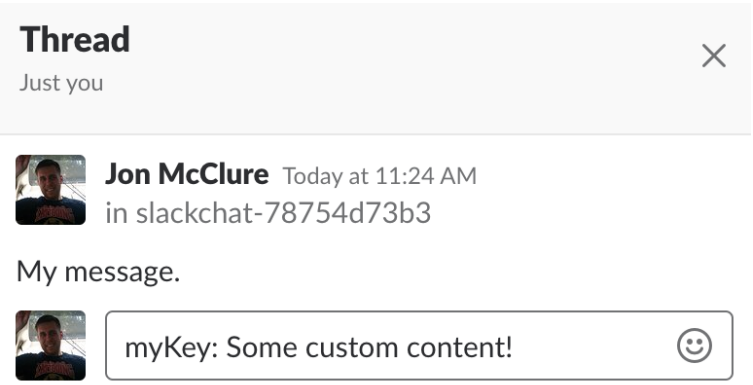

```
"messages": [
      {
          "timestamp": "2018-02-04T15:00:45.000065Z",
          "user": "SOMEUSER1",
          "content": "My message is ready to publish.",
          "args": ["edited"]
      },
  ]
```

You can also use a `CustomContentTemplate` instance to attach an arg to a message whenever the instance's `search_string` matches the content of a message.

kwargs

Kwargs are created by messages in a thread attached to a Slack message.

Create your threaded message with a key: value pair:



That pair will be parsed and serialized as kwargs on the message:

```
"messages": [
      {
          "timestamp": "2018-02-04T15:00:45.000065Z",
          "user": "SOMEUSER1",
          "content": "My message.",
          "kwargs": {
            "myKey": "Some custom content!"
          }
      },
  ]
```

One common use case for kwargs is to tag messages for use in custom navigation in the rendered slackchat.

Note: You can turn off kwarg handling for a `ChatType` by setting `kwargs_in_threads = False`.

5.1.8 Custom content templates

You can use a `CustomContentTemplate` to change the way messages' content is serialized or to add a custom arg to a message.

Set a regex `search_string` to match against messages' content and capture groups of any content you want to reformat. Then use one of the template features to customize your message.

Content

Add a `content_template` Python formatting string that will reformat content. Leaving this field blank will pass the entire message's contents through.

For example, you might set up a `CustomContentTemplate` instance like this:

```
# regex search string
template.search_string =  '^ALERT! (.*)'

# formatting string
template.content_template = '<span class="alert-bold">{0}</span>'
```

Now a message from Slack like this:

```
ALERT! New slackchat started!
```

. . . would be reformated in the serializer like this:

```
<span class="alert-bold">New slackchat started!</span>
```

Args

You can also add an `argument_template` to your template instance, which will place arguments in the matched message's `args` when serialized These arguments should be comma-separated and can be regular text or a Python formatted strings whose args are the capture groups matched by the search string.

For example . . .

```
# Message: ALERT red! New slackchat started!

# regex search string
template.search_string =  '^ALERT (.*)! (.*)'

# formatting string
template.content_template = '{1}'

# argument template
template.argument_template = 'alert, alert-{0}'
```

. . . would render like this in the serializer of a matched message:

```
"messages": [
      {
          "timestamp": "2018-02-04T15:00:45.000065Z",
          "user": "SOMEUSER1",
          "content": "New slackchat started!",
          "args": ["alert", "alert-red"]
```

(continues on next page)

(continued from previous page)

```
        },
    ]
```

Attachment

A custom attachment can be added to your message using a JSON object schema. The values of the object can be Python formatted strings once again passed the args of the capture group.

When creating a custom attachment you might consider consulting Slack's attachment documentation to keep some sense of consistency between Slack-generated attachments and your custom one.

For example . . .

```
# Message: ALERT red! New slackchat started!

# regex search string
template.search_string =  '^ALERT (.*)! (.*)'

# formatting string
template.content_template = '{1}'

# attachment template
template.attachment_template = {
        "title": "Alert!",
        "service" "Alerter"
        "title_link": "http://example.com/alert",
        "text": "{0}",
        "color": "#ff0000"
    }
```

. . . would render like this in the serializer of a matched message:

```
"messages": [
        {
            "timestamp": "2018-02-04T15:00:45.000065Z",
            "user": "SOMEUSER1",
            "content": "New slackchat started!",
            "attachments": [
                {
                    "title": "Alert!",
                    "service" "Alerter"
                    "title_link": "http://example.com/alert",
                    "text": "red",
                    "color": "#ff0000"
                }
            ]
        },
    ]
```

Kwargs

Custom kwargs are also available using a JSON object schema. The values of the object can be Python formatted strings once again passed the args of the capture group. If there are duplicate-key conflicts between these and kwargs added via message threads, the message threads will take precedence.

For example . . .

```
# Message: ALERT red! New slackchat started!

# regex search string
template.search_string =  '^ALERT (.*)! (.*)'

# formatting string
template.content_template = '{1}'

# kwarg template
template.kwarg_template = {
  'alert-type': '{0}'
}
```

. . . would render like this in the serializer of a matched message:

```
"messages": [
      {
          "timestamp": "2018-02-04T15:00:45.000065Z",
          "user": "SOMEUSER1",
          "content": "New slackchat started!",
          "kwargs": {
            "alert-type": "red"
          }
      },
  ]
```

It's up to you to make sure your regex search strings aren't too greedy.

CHAPTER 6

Webhooks

Slackchat-serializer will fire webhooks whenever `Message`, `Reaction`, `Attachment` or `KeywordArgument` objects are saved or deleted.

6.1 Verifying your endpoint

Slackchat asks to verify your endpoint before it will send notifications.

To do so, it will send a payload that looks like this:

```
{
  "token": "your-webhook-verification-token",
  "type": "url_verification",
  "challenge": "a-challenge",
}
```

You endpoint should sniff the `type` and reply back with the challenge.

For example, you could use Django Rest Framework in a view to respond to the challenge like this:

```
class Webhook(APIView):
    def post(self, request, *args, **kwargs):
        payload = request.data

        if payload.get('token') != WEBHOOK_VERIFICATION_TOKEN:
            return Response(status=status.HTTP_403_FORBIDDEN)

        if payload.get('type') == 'url_verification':
            return Response(
                data=payload.get('challenge'),
                status=status.HTTP_200_OK
            )
        # Now handle regular notifications...
```

Note: If you need to fire a repeat verification request because your endpoint didn't respond correctly the first time or because the endpoint URL changed, simply open the `Webhook` instance in Django's admin and re-save it.

6.2 Update Payload

Whenever one of the notification models is updated, the app will send a payload to every verified endpoint with the ID of the channel that was updated, allowing your renderer to hit the channel's API and republish the updated data.

```
{
  "token": "your-webhook-verification-token",
  "type": "update_notification",
  "channel": "a-channel-uuid-xxxx...",
  "chat_type": "a-chat-type"
}
```

If the type of the webhook is `update_notification`, the payload will also include an `update_type` of `message_created`, `message_changed`, or `message deleted`. It will also include a `message` key with data about the message that was added, changed, or deleted.

```
{
  "token": "your-webhook-verification-token",
  "type": "update_notification",
  "channel": "a-channel-uuid-xxxx...",
  "chat_type": "a-chat-type"

  "update_type": "message_added",
  "message": {
    "timestamp": "2018-10-16T17:23:49.000100Z",
    "user": "USERID",
    "content": "A new message."
  }
}
```

6.3 Explicit Payloads

Users can also use the Django admin or CMS to send endpoints explicit request payloads.

6.3.1 `republish_request`

A request to publish (or republish) all static assets associated with the channel. It carries with it the channel data as a serialized JSON string.

```
{
  token: "your-webhook-verification-token",
  type: "republish_request",
  channel: "a-channel-uuid-xxxx...",
  channel_data: "{ ... \"title\": \"Channel Title\", \"introduction\": \"Lorem ipsum\
↪", ... }",
  chat_type: "a-chat-type"
}
```

6.3.2 unpublish_request

A request to unpublish (or otherwise remove) all static assets associated with the channel. It carries with it the channel data as a serialized JSON string.

```
{
  token: "your-webhook-verification-token",
  type: "unpublish_request",
  channel: "a-channel-uuid-xxxx...",
  channel_data: "{ ... "title": "Channel Title", "introduction": "Lorem ipsum", ... }
↪",
  chat_type: "a-chat-type"
}
```

CHAPTER 7

Integrating with a renderer

Slackchat-serializer generally doesn't enforce any conventions as to how you render a slackchat on a custom page or within your CMS.

That said, we recommend that you **do not** let readers hit the serializer API directly, but rather respond to webhooks from the app by republishing the data to a JSON file on a static file server like Amazon Web Services S3.

7.1 CORS

If you need to allow cross-origin requests in your renderer, we recommend using django-cors-headers in your project.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search