django-wordpress-rest Documentation

Release 0.1.1

Jeff Sternberg, Observer Media

August 30, 2015

Django-wordpress-rest is a Django application that syncs content from a WordPress.com site to a Django site. This is done using the WordPress.com REST API. A separate copy of the content data is stored on the Django side, which allows for loose coupling and extensability.

CHAPTER 1

Contents

1.1 Quickstart

Getting started with django-wordpress-rest is simple. Follow these quick steps to start.

1.1.1 Installation

Install the module via pip:

```
pip install django-wordpress-rest
```

1.1.2 Django settings

Add `"wordpress"` to your `INSTALLED_APPS` Django setting:

```
INSTALLED_APPS = (
    # ...
    "wordpress",
    # ...
)
```

1.1.3 Database migration

Create the database tables that will persist the sync'd WordPress content:

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

1.1.4 Sync your WordPress site

Sync your WordPress content using the management command. The `<site_id>` can be found using the /me/sites WordPress API call.

This is useful for periodically updating the content with cron. See Load Options for more.

```
$ python manage.py load_wp_api <site_id>
```

1.2 Authentication

If you'd like to synchronize private content (such as drafts) from your WordPress site to Django, create an OAuth2 access token using the instructions provided by WordPress: https://developer.wordpress.com/docs/oauth2/

Add this token to your Django `settings.py` file. Use an environment variable to keep things secure:

```
WP_API_AUTH_TOKEN = os.getenv("WP_API_AUTH_TOKEN")
```

1.3 Load Options

The `load_wp_api` management command supports several extended options via command-line arguments.

1.3.1 Default (For Periodic Syncs)

The default, without any args except `site_id`, is designed to bring the site content up to date when it runs. It uses the modified dates of posts to load only "new" content.

For example:

```
# first run gets everything
$ python manage.py load_wp_api <site_id>

# second run gets content modified since previous run
$ python manage.py load_wp_api <site_id>
```

1.3.2 Full

To do a full sweep of the site content, inserting and updating as needed, use the `--full` argument:

```
# first run gets everything
$ python manage.py load_wp_api <site_id>

# second run gets/updates all content again
$ python manage.py load_wp_api <site_id> --full
```

1.3.3 Modified Date

You can also load everything modified after a given date with `--modified_after` argument:

```
$ python manage.py load_wp_api <site_id> --modified_after=2015-01-01
```

1.3.4 Content Type

To load only a single type of content, such as posts, pages, attachments, or reference data (authors, tags, categories, media):

```
$ python manage.py load_wp_api <site_id> --type=post
$ python manage.py load_wp_api <site_id> --type=page
$ python manage.py load_wp_api <site_id> --type=attachment
$ python manage.py load_wp_api <site_id> --type=ref_data
```

1.3.5 Purge and Reload

Purge local content before loading – *careful*, this is destructive!

```
$ python manage.py load_wp_api <site_id> --purge --full
```

1.4 Webhook

The webhook is designed to allow you to sync a post to Django immediatey after it's updated on the WordPress site.

This is helpful, for example, with corrections that need to be published as soon as possible. It also helps content authors preview their posts on the Django site more quickly.

1.4.1 urls.py

If you'd like to use the webhook to sync a post immediately after it's updated, include the `urls` into your project's `urls.py`, like so:

```
from django.conf.urls import include

urlpatterns = [
    url(r'^wordpress/', include('wordpress.urls'))
]
```

1.4.2 after_response library

Add `"after_response"` to your `INSTALLED_APPS` setting (this allows asynchronous processing):

```
INSTALLED_APPS = (
    # ...
    "after_response",
    "wordpress",
    # ...
)
```

1.4.3 Django settings

The webhook looks for your `<site_id>` in Django settings. So add this your `settings.py`, and use an environment variable to keep things secure:

```
WP_API_SITE_ID = os.getenv("WP_API_SITE_ID")
```

1.4.4 WordPress save_post action

Finally from your WordPress.com site, submit a POST request with an `ID` data element in the body to trigger a sync of a single post. Note this should be the WordPress Post ID, not the Django one!

Something like this in your `functions.php` should work (note, how you implement this depends on your WordPress install):

```php
<?php
/**
 * Notify Django when saving a post
 * @param int $post_id
 * @return void
 */
function django_webhook( $post_id ) {
    // don't do this for autosave
    if ( defined( 'DOING_AUTOSAVE' ) && DOING_AUTOSAVE ) {
        return;
    }

    // if the post is live, or going live soon, notify Django that we created/updated it
    $post_status = get_post_status( $post_id );
    if ( in_array( $post_status, array("draft", "future", "publish"), true ) ) {
        $params = array(
            'ID' => $post_id,
        );
        wp_remote_post( "http://mydjangosite.com/wordpress/load_post",
                        array( 'method' => 'POST', 'body' => $params ) );
    }
}
add_action( 'save_post', 'django_webhook' );
```

Here's an example with curl for testing purposes:

```
$ curl -X POST --data "ID=123456" http://mydjangosite.com/wordpress/load_post
```

1.5 Change Log

1.5.1 0.1

Initial version

1.5.2 0.1.1

Fixed unexpected kwarg in load_wp_post()

1.5.3 0.1.2

Read The Docs documentation