Basket Documentation

Release 1.0

Mozilla

Apr 24, 2023

CONTENTS

This documentation explains how to install and use basket.mozilla.org.

CHAPTER

ONE

ABOUT BASKET

A Python web service, basket, provides an API for all of our subscribing needs. Basket interfaces into whatever email provider we are using, currently ConTact Management System (CTMS) (formerly Salesforce Marketing Cloud, formerly ExactTarget).

CHAPTER

TWO

CONTENTS

2.1 Installing Basket

2.1.1 Requirements

- Docker
- Docker-compose

You can install Docker CE for your platform at https://docker.com.

2.1.2 Installation

Get the code

```
git clone git@github.com:mozmeao/basket.git
```

Settings

Settings are injected into the Docker container environment via the *.env* file. You can get started by copying `env-dist` to `.env` and that will provide the basics you need to run the site and the tests.

Git Hooks

Install pre-commit, and then run `pre-commit install` and you'll be setup to auto format your code according to our style and check for errors for every commit.

Use Docker

The steps to get up and running are these:

```
$ # this pulls our latest builds from the docker hub.
$ # it's optional but will speed up your builds considerably.
$ docker-compose pull
$ # this starts the server and dependencies
$ docker-compose up web
```

If you've made changes to the *Dockerfile* or *requirements/*.txt* you'll need to rebuild the image to run the app and tests:

```
$ docker-compose build web
```

Then to run the app you run the *docker-compose up web* command again, or for running tests against your local changes you run:

```
$ docker-compose run --rm test
```

We use pytest for running tests. So if you'd like to craft your own pytest command to run individual test files or something you can do so by passing in a command to the above:

```
$ docker-compose run --rm test py.test basket/news/tests/test_views.py
```

And if you need to debug a running container, you can open another terminal to your basket code and run the following:

```
$ docker-compose exec web bash
$ # or
$ docker-compose exec web python manage.py shell
```

Maintaining Python requirements

```
$ # If you've added a new dependency or changed the hard pinning of one
$ make compile-requirements
$ # or to just check if there are stale deps so you can
$ # update the hard pinning in the *.in files
$ make check-requirements
```

Install Python requirements locally

Ideally, do this in a virtual environment (eg a *venv* or *virtualenv*)

```
$ make install-local-python-deps
```

2.2 Newsletter API

This "news" app provides a service for managing Mozilla newsletters.

fixtures/newsletters.json is a fixture that can be used to load some initial data, but is probably out of date by the time you read this.

Currently available newsletters can be found in JSON format via the /news/newsletters/ API endpoint.

If 'token-required' is specified, a token must be suffixed onto the API URL, such as:

```
/news/user/<token>/
```

This is a user-specific token given away by the email backend or basket in some manner (i.e. emailed to the user from basket). This token allows clients to do more powerful things with the user.

A client might also have an `API key` that it can use with some APIs to do privileged things, like looking up a user from their email.

If 'SSL required', the call will fail if not called over a secure (SSL) connection.

Whenever possible (even when the HTTP status is not 200), the response body will be a JSON-encoded dictionary with several guaranteed fields, along with any data being returned by the particular call:

> 'status': 'ok' if the call succeeded, 'error' if there was an error

If there was an error, these fields will also be included:

> 'code': an integer error code taken from `basket.errors` in basket-client. 'desc': brief English description of the error.

The following URLs are available (assuming "/news" is app url):

2.2.1 /news/subscribe

This method subscribes the user to the newsletters defined in the "newsletters" field, which should be a comma-delimited list of newsletters. "email" and "newsletters" are required:

```
method: POST
fields: email, format, country, lang, newsletters, optin, source_url,␣
↪trigger_welcome, sync
returns: { status: ok } on success
         { status: error, desc: <desc>, code: <error_code> } on error
SSL required if sync=Y
token or API key required if sync=Y
```

`format` can be any of the following values: H, html, T, or text

`country` is the 2 letter country code for the subscriber.

`lang` is the language code for the subscriber (e.g. de, pt-BR)

`first_name` is the optional first name of the subscriber.

`last_name` is the optional last name of the subscriber.

`optin` should be set to "Y" if the user should not go through the double-optin process (email verification). Setting this option requires an API key and the use of SSL. Defaults to "N".

`trigger_welcome` should be set to "N" if you do not want welcome emails to be sent once the user successfully subscribes and verifies their email. Defaults to "Y".

`sync` is an optional field. If set to Y, basket will ensure the response includes the token for the provided email address, creating one if necessary. If you don't need the token, or don't need it immediately, leave off `sync` so Basket has the option to optimize by doing the entire subscribe in the background after returning from this call. Defaults to "N".

Using `sync=Y` requires SSL and an API key.

`source_url` is an optional place to add the URL of the site from which the request is being made. It's just there to give us a way of discovering which pages produce the most subscriptions.

If the email address is invalid (due to format, or unrecognized domain), the error code will be `BASKET_INVALID_EMAIL` from the basket client.

2.2.2 /news/unsubscribe

This method unsubscribes the user from the newsletters defined in the "newsletters" field, which should be a comma-delimited list of newsletters. If the "optout" parameter is set to Y, the user will be opted out of all newsletters. "email" and either "newsletters" or "optout" is required:

```
method: POST
fields: email, newsletters, optout
returns: { status: ok } on success
         { status: error, desc: <desc> } on error
token-required
```

2.2.3 /news/user

Returns information about the user including all the newsletters he/she is subscribed to:

```
method: GET
fields: *none*
returns: {
    status: ok,
    email: <email>,
    format: <format>,
    country: <country>,
    lang: <lang>,
    newsletters: [<newsletter>, ...]
} on success
{
    status: error,
    desc: <desc>
} on error
token-required
```

The email will be masked unless the request was made with a valid API key.

If POSTed, this method updates the user's data with the supplied fields. Note that the user is only subscribed to "newsletters" after this, meaning the user will be unsubscribed to all other newsletters. "optin" should be Y or N and opts in/out the user:

```
method: POST
fields: email, format, country, lang, newsletters, optin
returns: { status: ok } on success
         { status: error, desc: <desc> } on error
token-required
```

2.2.4 /news/newsletters

Returns information about all of the available newsletters:

```
method: GET
fields: *none*
returns: {
    status: ok,
    newsletters: {
        newsletter-slug: {
```

(continues on next page)

(continued from previous page)

```
                vendor_id: "ID_FROM_EXACTTARGET",
                welcome: "WELCOME_MESSAGE_ID",
                description: "Short text description",
                show: boolean,  // whether to always show this in lists
                title: "Short text title",
                languages: [
                    "<2 char lang>",
                    ...
                ],
                active: boolean,  // whether to show it at all (optional)
                order: 15,  // in what order it should be displayed in lists
                requires_double_optin: boolean
            },
            ...
    }
}
```

2.2.5 /news/debug-user

REMOVED. Will return a 404. Use the newer and better `lookup-user` method.

2.2.6 /news/lookup-user

This allows retrieving user information given either their token or their email (but not both). To retrieve by email, an API key is required:

```
method: GET
fields: token, or email and api-key
returns: { status: ok, user data } on success
         { status: error, desc: <desc> } on error
SSL required
token or API key required
```

Examples:

```
GET https://basket.example.com/news/lookup-user?token=<TOKEN>
GET https://basket.example.com/news/lookup-user?api-key=<KEY>&email=
↪<email@example.com>
```

The API key can be provided either as a GET query parameter `api-key` or as a request header `X-api-key`. If both are provided, the query parameter is used.

If user is not found, returns a 404 status and ‘desc’ is ‘No such user’.

On success, response is a bunch of data about the user:

```
{
    'status':  'ok',       # no errors talking to CTMS
    'status':  'error',    # errors talking to CTMS, see next field
    'desc':  'error message'    # details if status is error
    'email': 'email@address',
    'format': 'T'|'H',
    'country': country code,
    'lang': language code,
```

(continues on next page)

(continued from previous page)

```
    'token': UUID,
    'created-date': date created,
    'newsletters': list of slugs of newsletters subscribed to,
    'confirmed': True if user has confirmed subscription (or was excepted),
    'pending': True if we're waiting for user to confirm subscription
    'master': True if we found them in the master subscribers table
}
```

The email will be masked unless the request was made with a valid API key.

Note: Because this method always calls the backing contact management system one or more times, it can be slower than some other Basket APIs, and will fail if it is down.

2.2.7 /news/recover/

This sends an email message to a user, containing a link they can use to manage their subscriptions:

```
method: POST
fields: email
returns:  { status: ok } on success
          { status: error, desc: <desc> } on error
```

The email address is passed as 'email' in the POST data. If it is missing or not syntactically correct, a 400 is returned. Otherwise, a message is sent to the email, containing a link to the existing subscriptions page with their token in it, so they can use it to manage their subscriptions.

If the user is known in CTMS, the message will be sent in their preferred language and format.

If the email provided is not known, a 404 status is returned.