Basket Documentation

Release 1.0

Mozilla

November 13, 2015

Contents

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

CHAPTER 1

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.

CHAPTER 2

Contents

2.1 Installing Basket

2.1.1 Requirements

- Python >= 2.6, < 3
- MySQL

2.1.2 Installation

Get the code

```
git clone git@github.com:mozilla/basket.git --recursive
```

The *–recursive* is important!

Make a virtualenv

Using virtualenvwrapper:

```
mkvirtualenv --python=python2.6 basket
```

Install packages

```
pip install -r requirements/compiled.txt
```

For developers:

```
pip install -r requirements/dev.txt
```

Settings

Create a settings_local.py file. Typical settings can be found in settings_ex.py NOTE: make sure you have `from settings import *` at the top, or you'll be confused when things aren't working correctly.

Database schema

```
./manage.py syncdb --noinput
```

2.2 Production Environments

Production installs often have a few different requirements:

- point Apache's `WSGIScriptAlias` at `/path/to/basket/wsgi/basket.wsgi`
- jbalogh has a good example WSGI config for Zamboni.
- `DEBUG = False` in settings

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fixtures/newsletters.json is a fixture that can be used to load some initial data, but is probably out of date by the time you read this.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

2.3.1 /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)

`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. If it is likely just a misspelled domain, then basket may suggest a correction. If there is a suggestion, it will be in the error response in the `suggestion` parameter.

If you've validated that the email address is indeed correct and don't want the validation, you may pass `validated=true` with your submission and the address will be accepted. It is suggested that you not use this unless you've specifically asked the user if it is really correct.

2.3.2 /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
```

2.3.3 /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
```

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
```

2.3.4 /news/newsletters

Returns information about all of the available newsletters:

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

2.3.5 /news/debug-user

This is the same as a GET request to /user, except that you must pass in the email and a supertoken as GET params. The supertoken is a special token that should never be made public and lets devs debug users to make sure they were entered into the system correctly:

```
    method: GET
    fields: email, supertoken
```

2.3.6 /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 ET
    'status':  'error',   # errors talking to ET, see next field
    'desc':  'error message'   # details if status is error
    'email': 'email@address',
    'format': 'T'|'H',
    'country': country code,
    'lang': language code,
    '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
}
```

Note: Because this method always calls Exact Target one or more times, it can be slower than some other Basket APIs, and will fail if ET is down.

2.3.7 /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 ET, the message will be sent in their preferred language and format.

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