XmppFlask Documentation

Release 0.1

Konstantine Rybnikov

Sep 27, 2017

Contents

1 XmppFlask is easy to start with **3**

2 Source Code **5**

3 Status **7**

4 Authors **9**

5 Community **11**

6 Where should I go next? **13**

6.1 Introduction 13

6.2 Overview 15

6.3 Indices and tables 17

XmppFlask is easy to use XMPP framework that is inspired (heavily) by Flask. It is intended to be as easy to use as Flask itself is.

CHAPTER 1

XmppFlask is easy to start with

The main idea is to make you happy with writing small jabber bots. Like this:

```
from xmppflask import XmppFlask
app = XmppFlask(__name__)

@app.route(u'ping')
def ping():
    return u'pong'
```

CHAPTER 2

Source Code

Source code is available via bitbucket https://bitbucket.org/k_bx/xmppflask.

CHAPTER 3

Status

It's in status of ideal suitability for "use and help polishing it", since some obvious improvements could be done.

CHAPTER 4

Authors

Original author is Konstantine Rybnikov. Current main developer is Alexander Shorin

Great thanks to contributors:

- Brendan McCollam

Feel free to be the next one.

CHAPTER 5

Community

Join us at jabber conference xmppflask@conference.jabber.org for discussions. Homepage is located at http://xmppflask.org.

CHAPTER 6

Where should I go next?

You can go directly to *Introduction*.

Contents:

Introduction

XmppFlask is considered to be simple (it might be not there yet, but we are constantly working no it).

So let's begin with "hello world" app.

Installation

Note: We use PyPy interpreter to run things, but you may want to use standard CPython instead.

Installing pypy

Go to http://pypy.org , download latest pypy, unpack it somewhere and make

```
sudo ln -s /absolute/path/to/pypy/bin/pypy /usr/local/bin/pypy
```

Now create a virtual environment for you application. First install latest virtualenv to do that:

```
sudo easy_install --upgrade virtualenv
```

Then create an environment:

```
mkdir ~/v
virtualenv -p /usr/local/bin/pypy ~/v/ping-env
```

Now go ahead and activate that environment:

```
source ~/v/ping-env/bin/activate
```

From now on you can install things and they will only harm your virtual environment.

Installing XmppFlask

Right now the simplest way is just to clone the source repository

```
hg clone https://k_bx@bitbucket.org/k_bx/xmppflask
cd xmppflask
python setup.py install
```

Writing simple application

`cd` to your project's folder (like `~/workspace/pingpong` and write file called pingpong.py

```
# -*- coding: utf-8 -*-

from xmppflask import XmppFlask
app = XmppFlask(__name__)

@app.route(u'ping')
def ping():
    return u'pong'
```

This small app responses "pong" to every "ping" message it gets. Now we need to run that somehow.

If you have setuptools package installed the simplest way to run your XMPPWSGI app would be using *xmppflask* console script. The other way to run your app is `python path/to/xmppflask/run.py` - *xmppflask* is just handy shortcut.

```
xmppflask --jid xmppflask@example.com \
--password isecretlyusedjango ./pingpong.py:app
```

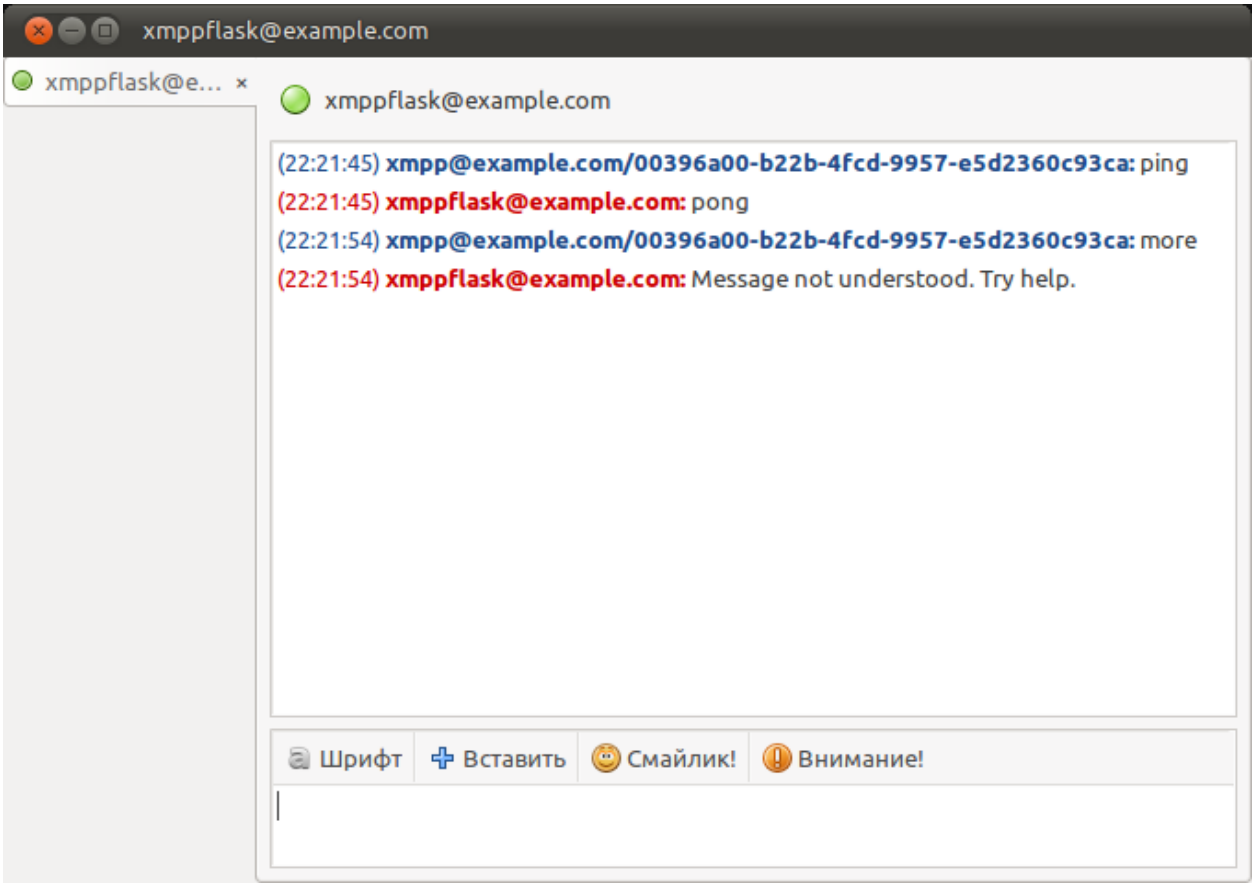
Note: You need to already have xmppflask@example.com jabber account with that strange password.

Warning: Passing JID and his password as command line arguments may be not very secure idea. As alternative solution, you could set them via `XMPPFLASK_JID` and `XMPPFLASK_PASSWORD` environment variables. Also you may skip this arguments - they will be asked to be prompted from tty.

Now run that and you should see something like this:

```
(xmpp)kost@kost-narwhal:~/workspace/pingpong$ xmppflask \
--jid xmppflask@example.com --password isecretlyusedjango ./pingpong.py:app
INFO:root:connecting...
INFO:root:done.
WARNING:root:Unable to establish secure connection - TLS failed!
INFO:root:> bot started!
```

Now go ahead and write something to our bot. Pidgin can be your friend!



Great! Now you can go to *Overview* page and see what else is there.

Overview

Basic idea behind XmppFlask is being as simlpe as Flask is. So let's start from a simple application.

```
# -*- coding: utf-8 -*-

from xmppflask import XmppFlask
app = XmppFlask(__name__)

@app.route(u'ping')
def ping():
    return u'pong'
```

Routing

Routing is pretty much the same as in Flask http://flask.pocoo.org/docs/quickstart/#routing:

```
@app.route(u'ping <user> <int:n> times')
def ping(user, n):
    return u'ponged %s %s times' % (user, n)
```

Rendering Templates

Take a look at http://flask.pocoo.org/docs/quickstart/#rendering-templates

```
@app.route(u'ping')
def ping():
    return render_template(u'ping.html')
```

Logging

http://flask.pocoo.org/docs/quickstart/#logging

```
app.logger.debug('A value for debugging')
app.logger.warning('A warning occurred (%d apples)', 42)
app.logger.error('An error occurred')
```

Sessions

You may keep session context for each user that interacts with your app. Just define session interface:

```
from xmppflask import XmppFlask
from xmppflask.session import MemorySessionInterface
from xmppflask import session

app = XmppFlask(__name__)
app.session_interface = MemorySessionInterface()

@app.route(u'ping')
def ping():
    if 'seq' not in session:
        session['seq'] = 0
    session['seq'] += 1
    return u"ping seq %d. PONG!" % session['seq']
```

And more

As in Flask you can do things like this:

```
@app.before_request
def before_request():
    g.db = connect_db()

@app.teardown_request
```

```
def teardown_request(exception):
    g.db.close()
```

XMPPWSGI

In XmppFlask there's a thing called XMPPWSGI. Basically it's a "WSGI for XMPP" (as you already might have guessed). To read about the whole WSGI idea you can go and read PEP 333. In XMPPWSGI, your application should be something like this (could be changed in near time):

```
def xmppwsgi_app(self, environ, notification_queue):
    notification_queue.append(
        [('user1@gmail.com', 'notification 1'),
         ('user2@gmail.com', 'notification 2')])
    return u'response to user'
```

This code is a simple XMPPWSGI app that responses to user by string "response to user" and also says to XMPPWSGI server to send messages to user1@gmail.com and user2@gmail.com.

Testing

Just run `nosetests` inside environment.

Indices and tables

- genindex
- modindex
- search