permabots Documentation

Release 2.2.3

Juan Madurga

June 07, 2016

Contents

With Permabots you can build chat bots and with the same configuration use it for several instant messaging providers. Permabots was born to be a microservice to connect messaging providers to your apps using REST APIs.

Contents:

CHAPTER 1

Permabots

CI: PyPI: Docs: Build chat bots and connect then to your app APIs.

With Permabots you can build chat bots and with the same configuration use it for several instant messaging providers. Permabots was born to be a microservice to connect messaging providers to your apps using REST APIs.

1.1 Documentation

The full documentation is at https://permabots.readthedocs.org.

1.2 Features

- Telegram, Kik and Facebook Messenger bots
- Message handling definition with regex, as django urls.
- HTTP methods: GET/POST/PUT/DELETE/PATCH
- Text responses and keyboards with Jinja2 templates
- Chat State handling
- Asynchronous processing of messages
- Media messages not supported

1.3 Quickstart

Install permabots:

```
pip install permabots
```

Add `permabots` to your `INSTALLED_APPS`, and run:

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

Instant messaging providers uses webhooks to send messages to your bots. Add permabots processing urls to your urlpatterns:

```
url(r'^processing/', include('permabots.urls_processing', namespace="permabots"))
```

Webhooks are generated with `django.contrib.sites`. You need it installed and `SITE_ID` configured. If you want to generate webhook manually you can do it:

```
MICROBOT_WEBHOOK_DOMAIN = 'https://yourdomain.herokuapp.com'
```

It is usefull when you don't have `https` in your public domain but you have it in your autogenerated domain. i.e. heroku.

Bots are associated to Django Users. You need at least one user, for example admin user.

Then you can create all permabots data, Bots, Conversation Handlers, Notitication Hooks,... via Django admin or with REST API (recommended).

Remember there is an online implementation in www.permabots.com to use it for free.

1.4 Demo

You can check and deploy a Permabots demo https://github.com/jlmadurga/permabots-demo

1.5 Running Tests

Does the code actually work?

```
source <YOURVIRTUALENV>/bin/activate
(myenv) $ pip install -r requirements/test.txt
(myenv) $ make test
(myenv) $ make test-all
```

CHAPTER 2

Installation

At the command line:

```
$ pip install permabots
```

Add `permabots` to your `INSTALLED_APPS`, and run:

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

Instant messaging providers uses webhooks to send messages to your bots. Add permabots processing urls to your urlpatterns:

```
url(r'^processing/', include('permabots.urls_processing', namespace="permabots"))
```

Webhooks are generated with `django.contrib.sites`. You need it installed and `SITE_ID` configured. If you want to generate webhook manually you can do it:

```
MICROBOT_WEBHOOK_DOMAIN = 'https://yourdomain.herokuapp.com'
```

Tip: It is usefull when you don't have `https` in your public domain but you have it in your autogenerated domain. i.e. heroku.

Bots are associated to Django Users. You need at least one user, for example the admin user.

2.1 Demo

You can check or deploy this Permabots demo https://github.com/jlmadurga/permabots-demo

2.2 REST API

Permabots comes with a REST API to manage all elements. If you want to use REST API *(recommended)* permabots requires `rest_framework` and `rest_framework.authtoken`. Added them to your `INSTALLED_APPS`.

And then configure urls:

```
url(r'^api/v1/', include('permabots.urls_api', namespace="api"))
```

API uses Token authentication. Generate token when user is created:

```
@receiver(post_save, sender=settings.AUTH_USER_MODEL)
def create_auth_token(sender, instance=None, created=False, **kwargs):
if created:
        Token.objects.create(user=instance)
```

Note: You can check API in official Permabots.com implementation.

2.3 Celery

Permabots uses celery to delegate processing to workers and release webhooks. Permabots do not require any special settings for celery.

Tip: Just remember using DB as broker will slow down the processing. Do not used in order to avoid delay in bot replays.

2.4 Cache

Cache is extensively used by Permabots. Set your preferred cache backend to reduce processing time.

Note: Once a chat bot is configured most of their models are static `handlers`, `templates`, etc so it is not required to access DB each message arrives to permabots

CHAPTER 3

Usage

With Permabots you can build chat bots and with the same configuration use it for several instant messaging providers. Permabots was born to be a microservice to connect messaging providers to your apps using REST APIs.

A bot in Permabots defines how to react to each client messages. To build responses the bot works with a context to generate responses with Jinja2 templates. This bot behavior can be shared by several instant messaging providers, saving time.

Note: Permabots django app do not have a dashboard. You can configure bots with the REST API (recommended) or using django admin. Remember there is an online implementation with dashboard in Permabots.com you can use it for free.

3.1 Bots

class permabots.models.bot.**Bot**(**args*, ***kwargs*)

Bases: permabots.models.base.PermabotsModel

Model representing a Permabot. Its behavior is shared by all service integrations.

Parameters

- **id** (*UUIDField*) – Id
- **created_at** (*DateTimeField*) – Date created
- **updated_at** (*DateTimeField*) – Date updated
- **owner_id** (*ForeignKey*) – User who owns the bot
- **name** (*CharField*) – Name for the bot
- **telegram_bot_id** (*OneToOneField*) – Telegram Bot
- **kik_bot_id** (*OneToOneField*) – Kik Bot
- **messenger_bot_id** (*OneToOneField*) – Messenger Bot

handle_hook(*hook*, *data*)

Process notification hook.

Parameters

- **hook** (Hook *Hook*) – Notification hook to process
- **data** – JSON data from webhook POST

handle_message(*message*, *bot_service*)
Process incoming message generating a response to the sender.

Parameters

- **message** – Generic message received from provider
- **bot_service** (IntegrationBot *IntegrationBot*) – Service Integration

Note: Message content will be extracted by IntegrationBot

class permabots.models.bot.**IntegrationBot**(**args*, ***kwargs*)
Bases: permabots.models.base.PermabotsModel

Abstract class to integrate new instant messaging service.

Parameters

- **id** (*UUIDField*) – Id
- **created_at** (*DateTimeField*) – Date created
- **updated_at** (*DateTimeField*) – Date updated
- **enabled** (*BooleanField*) – Enable/disable telegram bot

build_keyboard(*keyboard*)
From an arrays of strings generated specific keyboard for integration

Parameters **keyboard** – list(strings)

Returns specific keyboard

create_chat_state(*message*, *target_state*, *context*)
Crate specific chat state modelling for the integration. It is called only when first chat interaction is performed by a user.

Parameters

- **message** – Message from the provider
- **target_state** – State to set
- **context** – Processing generated in the processing

get_chat_id(*message*)
Extract chat identifier from service message.

Parameters **message** – Message from provider

Returns chat identifier

get_chat_state(*message*)
Each integration has its own chat state model. Implement this method to obtain it from message

Parameters **message** – Message from provider

Returns generic chat state

hook_id
Identifier to generate webhook url i.e. primary key UUID

Returns Identifier

Return type string

hook_url
Name of the view to resolve url. i.e. permabots:telegrambot :returns: Named view

identity
Some service identifier to attach in processing context i.e. telegram.

Returns Service Indentifier

Return type string

init_bot()
Implement this method to perform some specific intialization to the bot

message_text(*message*)
Extract text message from generic message :param message: Message from provider :returns: text from message :rtype: string

null_url
Return a none URL to remove webhook. i.e.: None

Returns None url

Note: Some providers API accepts None but others need a real url. Use https://example.com in this case

send_message(*chat_id*, *text*, *keyboard*, *reply_message=None*, *user=None*)
Send message with the a response generated.

Parameters

- **chat_id** – Identifier for the chat
- **text** – Text response
- **keyboard** – Keyboard response
- **reply_message** – Message to reply
- **user** – When no replying in some providers is not enough with chat_id

Note: Each provider has its own limits for texts and keyboards buttons. Implement here how to split a response to several messages.

set_webhook(*url*)
Implement this method set webhook if the services requires

Parameters **url** – URL generated to use for this bot

3.1.1 Telegram

Integrate to Telegram platform only requires `token` obtained when creating a bot with BotFather

class permabots.models.bot.**TelegramBot**(**args*, ***kwargs*)
Bases: *permabots.models.bot.IntegrationBot*

Telegram integration.

Permabots only requires token to set webhook and obtain some bot info.

Follow telegram instructions to create a bot and obtain its token https://core.telegram.org/bots#botfather.

Parameters

- **id** (*UUIDField*) – Id
- **created_at** (*DateTimeField*) – Date created
- **updated_at** (*DateTimeField*) – Date updated
- **enabled** (*BooleanField*) – Enable/disable telegram bot
- **token** (*CharField*) – Token provided by Telegram API https://core.telegram.org/bots
- **user_api_id** (*OneToOneField*) – Telegram API info. Automatically retrieved from Telegram

class permabots.views.hooks.telegram_hook.**TelegramHookView**(***kwargs*)
Bases: rest_framework.views.APIView

View for Telegram webhook

post(*request*, *hook_id*)

Process Telegram webhook.

1. Serialize Telegram message
2. Get an enabled Telegram bot
3. Create Update 5. Delay processing to a task 6. Response provider

3.1.2 Kik

Kik platform needs username and api-key. Visit Kik for more information.

class permabots.models.bot.**KikBot**(**args*, ***kwargs*)
Bases: *permabots.models.bot.IntegrationBot*

Kik integration.

Permabots sets webhook. Only requires api_key and username from Kik provider.

Follow Kik instructons to create a bot and obtain username and api_key https://dev.kik.com/.

Parameters

- **id** (*UUIDField*) – Id
- **created_at** (*DateTimeField*) – Date created
- **updated_at** (*DateTimeField*) – Date updated
- **enabled** (*BooleanField*) – Enable/disable telegram bot
- **api_key** (*CharField*) – Kik bot api key
- **username** (*CharField*) – Kik bot user name

class permabots.views.hooks.kik_hook.**KikHookView**(***kwargs*)
Bases: rest_framework.views.APIView

View for Kik webhook.

post(*request*, *hook_id*)

Process Kik webhook:

1. Get an enabled Kik bot
2. Verify Kik signature

3. Serialize each message
4. For each message create `KikMessage` and `KikUser`
5. Delay each message processing to a task
6. Response provider

3.1.3 Facebook Messenger

Permabots only require `Page Access Token` but webhook can't be set automatically. Facebook Messenger requires to set it in its dev dashboard. Permabots generates webhook with bot integration id which is `uuid`. This url completed with your domain will be used as the webook `https://yourdomain/processing/messengerbot/permabots_messenger_bot_id/` When Facebook requires a `Verify Token` for validate webhook use bot integration id.

Note: Bots integrations can be enable/disable whenever to avoid processing

---

class permabots.models.bot.**MessengerBot**(**args*, ***kwargs*)

Bases: *permabots.models.bot.IntegrationBot*

Facebook Messenger integration.

Permabots only uses Page Access Token but webhook is not set. It mus be set manually in Facebook dev platform using UUID generated as id of the messenger bot after creation in Permabots.

This bot is used to Verify Token and generate url https://domain/processing/messengerbot/permabots_messenger_bot_id/

Read Messenger documentation <*https://developers.facebook.com/docs/messenger-platform/quickstart*> _.

Parameters

- **id** (*UUIDField*) – Id
- **created_at** (*DateTimeField*) – Date created
- **updated_at** (*DateTimeField*) – Date updated
- **enabled** (*BooleanField*) – Enable/disable telegram bot
- **token** (*CharField*) – Messenger token

class permabots.views.hooks.messenger_hook.**MessengerHookView**(***kwargs*)

Bases: rest_framework.views.APIView

View for Facebook Messenger webhook

get(*request*, *hook_id*)

Verify token when configuring webhook from facebook dev.

MessengerBot.id is used for verification

post(*request*, *hook_id*)

Process Messenger webhook. 1. Get an enabled Messenger bot 3. For each message serialize 4. For each message create `MessengerMessage` 5. Delay processing of each message to a task 6. Response provider

3.2 Context

Context is used to parametrize processing and to render templates. Some context data is available just bots starts its processing and other is attached as a result of the processing.

3.3 Environment

Each bot can have an environment to generate key/value variables to use them accross all processing context. Environement is attached in context as `env`

Tip: Imagine you are using some auth token to build API requests, set token in environment instead of copying in every place you need it.

class `permabots.models.environment_vars.`**`EnvironmentVar`**(**args*, ***kwargs*)

Bases: `permabots.models.base.PermabotsModel`

Environment Variable associated to a Bot.

Use it in contexts as {{ env.variable_key }}.

Parameters

- **`id`** (*`UUIDField`*) – Id
- **`created_at`** (*`DateTimeField`*) – Date created
- **`updated_at`** (*`DateTimeField`*) – Date updated
- **`bot_id`** (*`ForeignKey`*) – Bot which variable is attached.
- **`key`** (*`CharField`*) – Name of the variable
- **`value`** (*`CharField`*) – Value of the variable

3.4 States

Permabots works generating responses depending on the input, but this is not enough. A bot can be modelled as a state machine and depending on its actual state handle messages. Several states can be defined for each bot.

Permabots maintains associations between chats and states.

Context used in the state transition is attached to the state. Previous state context can be accessed with `state_context` variable.

Tip: This is useful for example when making some kind of form of questionary to store responses.

class `permabots.models.state.`**`State`**(**args*, ***kwargs*)

Bases: `permabots.models.base.PermabotsModel`

Represents a state for a conversation and a bot.

Depending the state of the chat only some actions can be performed.

Parameters

- **id** (*UUIDField*) – Id
- **created_at** (*DateTimeField*) – Date created
- **updated_at** (*DateTimeField*) – Date updated
- **name** (*CharField*) – Name of the state
- **bot_id** (*ForeignKey*) – Bot which state is attached to

3.5 Conversation Handlers

A Conversation Handler defines how to react to an incoming message.

- Pattern: Regular expression the incoming message must fullfil to be executed
- Request: An HTTP request
- Response: Templates to generate text and keyboard response
- Source States: Bot must be in one of the states of this list to be executed
- Target State: State the bot transitionates when its is processed successfully
- Priority: It is usefull when more than handler fullfil conditions to be executed

class permabots.models.handler.**Handler**(**args*, ***kwargs*)
Bases: permabots.models.base.PermabotsModel

Model to handler conversation message

Parameters

- **id** (*UUIDField*) – Id
- **created_at** (*DateTimeField*) – Date created
- **updated_at** (*DateTimeField*) – Date updated
- **bot_id** (*ForeignKey*) – Bot which Handler is attached to
- **name** (*CharField*) – Name for the handler
- **pattern** (*CharField*) – Regular expression the Handler will be triggered. Using https://docs.python.org/2/library/re.html#regular-expression-syntax
- **request_id** (*OneToOneField*) – Request the Handler processes
- **response_id** (*OneToOneField*) – Template the handler uses to generate response
- **enabled** (*BooleanField*) – Enable/disable handler
- **target_state_id** (*ForeignKey*) – This state will be set when handler ends processing
- **priority** (*IntegerField*) – Set priority execution. Higher value higher priority

process(*bot*, *message*, *service*, *state_context*, ***pattern_context*)
Process conversation message.

1. **Generates context**
   - service: name of integration service
   - state_context: historic dict of previous contexts. identified by state
   - pattern: url pattern dict

- env: dict of environment variables associated to this bot
- message: provider message
- emoji: dict of emojis use named notation with underscores <*http://apps.timwhitlock.info/emoji/tables/unicode*> _.

2.Process request (if required)

3.Generates response. Text and Keyboard

4.Prepare target_state and context for updating chat&state info

Parameters

- **bot** – Bot the handler belongs to
- **message** – Message from provider
- **service** (*string*) – Identity integration
- **state_context** (*dict*) – Previous contexts
- **pattern_context** (*dict*) – Dict variables obtained from handler pattern regular expression.

Returns Text and keyboard response, new state for the chat and context used.

3.6 Notification Hooks

Notification hooks can be used to send messages to clients. Just define a response to be generated by POST data and a list of recipients for different providers.

class permabots.models.hook.**Hook**(**args*, ***kwargs*)

Bases: permabots.models.base.PermabotsModel

Notification Hook representation.

The webhook url is generated with the key.

Parameters

- **id** (*UUIDField*) – Id
- **created_at** (*DateTimeField*) – Date created
- **updated_at** (*DateTimeField*) – Date updated
- **bot_id** (*ForeignKey*) – Bot which Hook is attached
- **name** (*CharField*) – Name of the hook
- **key** (*CharField*) – Key generated to complete the Hook url. http://permabots.com/process/hook/{{key}}
- **response_id** (*OneToOneField*) – Template the hook uses to generate the response
- **enabled** (*BooleanField*) – Enable/disable hook

process(*bot*, *data*)

Notification hook processing generating a response.

Parameters

- **bot** – Bot receiving the hook

- **data** – JSON data from hook POST

Type JSON

class permabots.views.hooks.permabots_hook.**PermabotsHookView**(**kwargs)
Bases: rest_framework.views.APIView

View for Notification Hooks.

post(*request*, *key*)

Process notitication hooks:

1. Obtain Hook
2. Check Auth
3. Delay processing to a task
4. Respond requester

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/jlmadurga/permabots/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

4.1.4 Write Documentation

Permabots could always use more documentation, whether as part of the official permabots docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/jlmadurga/permabots/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *permabots* for local development.

1. Fork the *permabots* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/permabots.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv permabots
$ cd permabots/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 permabots tests
$ make tests
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/jlmadurga/permabots/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python runtests.py tests.test_functional
```

CHAPTER 5

Credits

5.1 Development Lead

- Juan Madurga <jlmadurga@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.0 (2016-05-16)

- First release on PyPI.

p