django-telegram-bot Documentation

Release 0.6.0

Juan Madurga

December 21, 2016

Contents

Contents:

CHAPTER 1

django-telegram-bot

CI: PyPI: Docs: Django app to write Telegram bots. Just define commands and how to handle them.

Try Permabots: more stable django app for bots https://github.com/jlmadurga/permabots

NOTE: Just for text messages at this moment.

1.1 Documentation

The full documentation is at https://django-telegram-bot.readthedocs.org.

Telegram API documentation at https://core.telegram.org/bots/api

1.2 Quickstart

Install django-telegram-bot:

```
pip install django-telegram-bot
```

Add `telegrambot` and `rest_framework` to your `INSTALLED_APPS`, and run:

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

After creating a bot in Telegram Platform, create at least one bot with django admin. Token is the only required field. You may need to provided public key certificate for your server. https://core.telegram.org/bots/self-signed Heroku has https and ssl by default so it is a good option if you dont want to deal with that.

Add webhook url to your urlpatterns:

```
url(r'^telegrambot/', include('telegrambot.urls', namespace="telegrambot")),
```

Define the file where commands will be defined in `urlpatterns` variable, analogue to django `urls` and `ROOT_URLCONF`:

```
TELEGRAM_BOT_HANDLERS_CONF = "app.handlers"
```

Set bot commands handlers is very easy just as defining *urls* in django. Module with `urlpatterns` that list different handlers. You can *regex* directly or use shortcuts like *command* or *unknown_command*

```
urlpatterns = [command('start', StartView.as_command_view()),
               command('author', AuthorCommandView.as_command_view()),
               command('author_inverse', AuthorInverseListView.as_command_view()),
               command('author_query', login_required(AuthorCommandQueryView.as_command_view())),
               unknown_command(UnknownView.as_command_view()),
               regex(r'author_(?P<name>\w+)', AuthorName.as_command_view()),
              ]
```

To set the webhook for telegram you need `django.contrib.sites` installed, `SITE_ID` configured in settings and with it correct value in the DB. The webhook for each bot is set when a Bot is saved and `enabled` field is set to true.

Bot views responses with Telegram messages to the user who send the command with a text message and keyboard. Compound with a context and a template. The way it is handled is analogue to Django views. Visits docs for more details https://django-telegram-bot.readthedocs.io/en/latest/usage.html

1.3 Features

- Multiple bots
- Message handling definition.
- Authentication
- Text responses and keyboards.
- Media messages not supported.
- Only Markup parse mode.

1.4 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 django-telegram-bot
```

Add `telegrambot` and `rest_framework` to `INSTALLED_APPS`:

```
INSTALLED_APPS=[
        ...
        "rest_framework",
        "telegrambot",
        ...
]
```

Migrate DB:

```
python manage.py migrate
```

CHAPTER 3

Usage

First you need a telegram bot and its token, visit https://core.telegram.org/bots.

After creating a bot in Telegram Platform, create at least one bot with django admin. Token is the only required field. You may need to provided public key certificate for your server. https://core.telegram.org/bots/self-signed Heroku has https and ssl by default so it is a good option if you dont want to deal with that. Add webhook url to your urlpatterns:

```
url(r'^telegrambot/', include('telegrambot.urls', namespace="telegrambot")
```

Define whe file where bot views will be defined in `urlpatterns` variable, analogue to django `urls` and `ROOT_URLCONF`:

```
TELEGRAM_BOT_HANDLERS_CONF = "app.handlers"
```

Set bot views handlers is very easy just as defining *urls* in django. Module with `urlpatterns` that list different handlers:

```
urlpatterns = [command('start', StartView.as_command_view()),
               command('author', AuthorCommandView.as_command_view()),
               command('author_inverse', AuthorInverseListView.as_command_view()),
               command('author_query', login_required(AuthorCommandQueryView.as_command_view())),
               unknown_command(UnknownView.as_command_view()),
               regex(r'author_(?P<name>\w+)', AuthorName.as_command_view()),
              ]
```

Set bot views handlers is very easy just as defining *urls'in django. Module with ''bothandlers'* list of different handlers *command('command', command_view)*, *regex('re_expresion', command_view)*,...:

```
bothandlers = [command('start', StartView.as_command_view())]
```

To set the webhook for telegram you need `django.contrib.sites` installed, `SITE_ID` configured in settings and with it correct value in the DB. The webhook for each bot is set when a Bot is saved and `enabled` field is set to true.

Bot views responses with Telegram messages to the user with a text message and keyboard. Compound with a context and a template. The way it is handled is analogue to Django views.

Define a bot view is really easy using generic classed views, analogues to django generic views.

Simple view just with a template, image /start command just to wellcome:

```
class StartView(TemplateCommandView):
        template_text = "bot/messages/command_start_text.txt"
```

List and detail views:

```
    class AuthorListView(ListCommandView):
    template_text = "bot/messages/command_author_list_text.txt"
    template_keyboard = "bot/messages/command_author_list_keyboard.txt"
    model = Author
    context_object_name = "authors"
    ordering = "-name"

class AuthorDetailView(DetailCommandView):
    template_text = "bot/messages/command_author_detail_text.txt"
    template_keyboard = "bot/messages/command_author_detail_keyboard.txt"
    context_object_name = "author"
    model = Author
    slug_field = 'name'
```

Most common use of commands is to have `/command` with no args for getting list and `/command element` for getting detail of one concrete element. It is easy to define:

```
class AuthorCommandView(ListDetailCommandView):
    list_view_class = AuthorListView
    detail_view_class = AuthorDetailView
```

Templates works just as normal django app. In /start command example it will search in templates dirs for `bot/messages/command_start_text.txt` to compound response message and `bot/messages/command_start_keyboard.txt`.

3.1 Authentication

If you require to be authenticated to perform some commands you can decorate `bot_views` with `login_required`. This is the flow the user will experience until being able to execute protected command:

- If chat is not already authenticated a message with a web link will be returned to login through the web site.
- Once logged, a link to open new authenticated chat will be returned to the user with deep linking mechanism.
- The user starts this new chat and now the bot will identify this chat as authenticated until token expires.

Define in `settings` the time life of a token:

```
TELEGRAM_BOT_TOKEN_EXPIRATION = 2 # two hours for a token to expire
```

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/django-telegram-bot/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

django-telegram-bot could always use more documentation, whether as part of the official django-telegram-bot 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/django-telegram-bot/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 *django-telegram-bot* for local development.

1. Fork the *django-telegram-bot* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django-telegram-bot.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 django-telegram-bot
$ cd django-telegram-bot/
$ 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:

```
    $ pip install -r requirements/dev.txt
$ make lint
$ pip install -r requirements/test.txt
$ make test
$ make test all
```

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, adn Python3. Check https://travis-ci.org/jlmadurga/django-telegram-bot/pull_requests and make sure that the tests pass for all supported Python versions.

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 (2016-21-01)

- First release on PyPI.