django-private-chat Documentation

Release 0.3.0

delneg

Jun 06, 2020

CONTENTS

1 :sunglasses: django-private-chat :sunglasses: 3
- 1.1 Important Notes . . . 3
- 1.2 Documentation . . . 4
- 1.3 Example project . . . 4
- 1.4 Customize the templates . . . 4
- 1.5 Exsiting project quickstart . . . 4
- 1.6 Running Tests . . . 6
- 1.7 Credits . . . 6

2 Installation 7

3 Usage 9

4 Messages 11

5 Settings 13

6 Admin 15

7 Starting the server 17

8 Contributing 19
- 8.1 Types of Contributions . . . 19
- 8.2 Get Started! . . . 20
- 8.3 Pull Request Guidelines . . . 21
- 8.4 Tips . . . 21

9 Credits 23
- 9.1 Development Lead . . . 23
- 9.2 Contributors . . . 23

10 0.3.0 (2020-05-03) 25

11 0.2.2 (2018-12-12) 27

12 0.2.1 (2018-12-07) 29

13 0.2.0 (2018-10-22) 31

14 0.1.9 (2018-07-16) 33

15 0.1.8 (2018-03-23) 35

16 0.1.7 (2018-03-20) 37

17 0.1.6 (2017-04-11) 39

18 0.1.5 (2017-03-11) 41

19 0.1.4 (2017-02-12) 43

20 0.1.3 (2017-02-11) 45

21 0.1.2 (2017-02-11) 47

22 0.1.1 (2017-02-10) 49

23 0.1.0 (2017-02-10) 51

Contents:

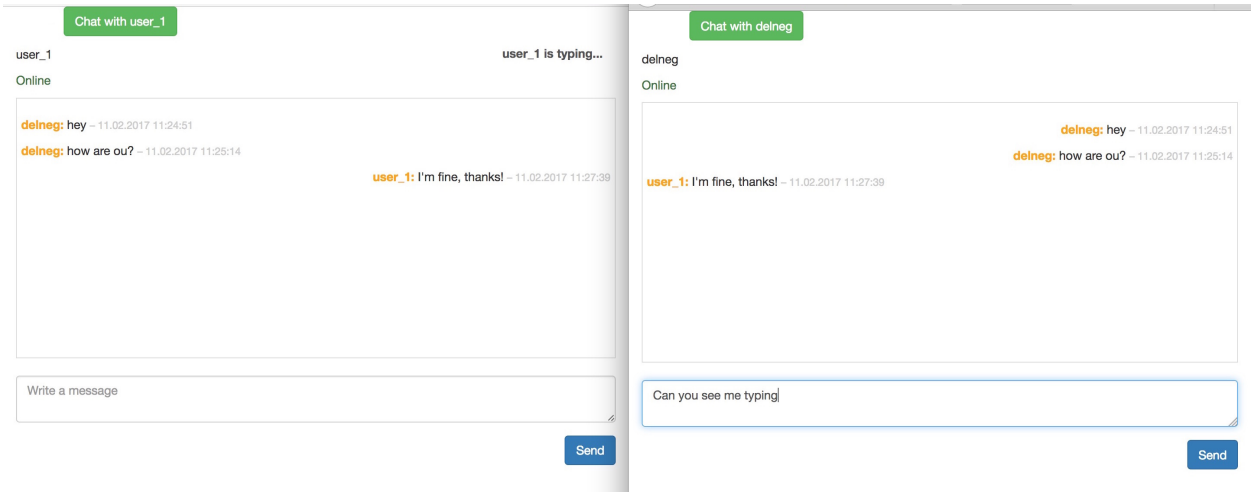
CHAPTER

ONE

:SUNGLASSES: DJANGO-PRIVATE-CHAT :SUNGLASSES:

Please also check out our another package https://github.com/Bearle/django_mail_admin

Django one-to-one Websocket-based Asyncio-handled chat, developed by Bearle team



1.1 Important Notes

This app uses separate management command, run_chat_server for running Websockets in Django context. It is intended to be used with something like Supervisor or Systemd to run asyncio webserver as a separate one from Django. We didn't want our app to be limited to be used together with Django Channels - that's why we did it that way.

You can find an example Systemd config to run it as a service at https://github.com/Bearle/django-private-chat/blob/dev/example.service

P.S. Don't forget to change CHAT_WS_SERVER_HOST && CHAT_WS_SERVER_PORT && CHAT_WS_SERVER_PROTOCOL settings!

1.2 Documentation

The full documentation is (finally) at https://django-private-chat.readthedocs.io . You can also check the docstrings & this readme.

1.3 Example project

You can check out our example project by cloning the repo and heading into example/ directory. There is a README file for you to check, initial data to check out the chat included.

1.4 Customize the templates

How to customize the template? Just copy:

```
venv/lib/pythonX.X/site-packages/django_private_chat/templates/django_private_chat/
↪dialogs.html
to
yourapp/templates/django_private_chat/dialogs.html
```

And feel free to edit it as you like! We intentionally left the JS code inside for it to be editable easily.

1.5 Exsiting project quickstart

Install django-private-chat:

```
pip install django-private-chat
```

Migrate:

```
python manage.py migrate django_private_chat
```

Note: you can use this package with or without uvloop, just run either

```
python manage.py run_chat_server
```

or run

```
python manage.py run_chat_server_uvloop
```

Add it to your *INSTALLED_APPS*:

```
INSTALLED_APPS = (
    ...
    'django_private_chat',
    ...
)
```

Add the server & port for your asyncio server to settings:

```
CHAT_WS_SERVER_HOST = 'localhost'
CHAT_WS_SERVER_PORT = 5002
CHAT_WS_SERVER_PROTOCOL = 'ws'
```

It is possible to change messages datetime format using

```
DATETIME_FORMAT
```

Add django-private-chat's URL patterns:

```
from django_private_chat import urls as django_private_chat_urls


urlpatterns = [
    ...
    url(r'^', include('django_private_chat.urls')),
    ...
]
```

Add

```
{% block extra_js %}{% endblock extra_js %}
```

to your base template

Now you can start a dialog using

```
/dialogs/some_existing_username
```

To create a WSS (TLS) server instead:

```
python manage.py run_chat_server "path/to/cert.pem"
```

(also works with uvloop). The "cert.pem" file should be a plaintext PEM file containing first a private key, then a certificate (may be a concatenation of a .key and a .crt file). Please note that wss will use TLSv1 by default for python 3.5 & 3.4 and will use ssl.PROTOCOL_TLS_SERVER for 3.6 and above. Features ———

-:white_check_mark: Uses current app model (get_user_model() and settings.AUTH_USER_MODEL)

-:white_check_mark: Translatable (uses ugettext and {% trans %})

-:white_check_mark: One-to-one user chat

-:white_check_mark: Works using WebSockets

-:white_check_mark: Works (optionally) using WSS (TLS) connections (disclaimer - security not guaranteed)

-:white_check_mark: Displays online/offline status

-:white_check_mark: Display typing/not typing status

-:white_check_mark: Soft deletable message model - be sure to keep messages to comply with message-keeping laws

-:white_check_mark: Flash the dialog button when the user you are not currently talking to wrote you a message

-:point_right: TODO: add a dialog to the list when new one started

-:point_right: TODO: add user-not-found and other alerts

-:point_right: possible Redis backend intergration

1.6 Running Tests

Does the code actually work?

```
source <YOURVIRTUALENV>/bin/activate
(myenv) $ pip install tox
(myenv) $ tox
```

1.7 Credits

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage

CHAPTER

TWO

INSTALLATION

At the command line:

```
$ pip install django-private-chat
```

CHAPTER

THREE

USAGE

To use django-private-chat in a project, add it to your *INSTALLED_APPS*:

```
INSTALLED_APPS = (
    ...
   'django_private_chat',
    ...
)
```

Add the server & port for your asyncio server to settings:

```
CHAT_WS_SERVER_HOST = 'localhost'
CHAT_WS_SERVER_PORT = 5002
CHAT_WS_SERVER_PROTOCOL = 'ws'
```

Add django-private-chat's URL patterns:

```
from django_private_chat import urls as django_private_chat_urls


urlpatterns = [
    ...
    url(r'^', include(django_private_chat_urls)),
    ...
]
```

or

```
urlpatterns = [
    ...
    path('', include(django_private_chat.urls)),
    ...
]
```

Add

```
{% block css %}{% endblock css %}
{% block content %}{% endblock content %}
{% block extra_js %}{% endblock extra_js %}
```

to your base template

Migrate:

```
python manage.py migrate django_private_chat
```

Now start the chat server:

```
python manage.py run_chat_server
```

CHAPTER

FOUR

MESSAGES

Application provides the following message channels:

```
'new-message',
'new-user',
'online',
'offline',
'check-online',
'is-typing',
'read_message'
```

which are pretty self-explanatory.

Here is detailed explanation of what each channel does and data types:

new-message Example from js:

```
{
    type: 'new-message',
    session_key: '{{ request.session.session_key }}',
    username: opponent_username,
    message: message
}
```

In the handler, a new Message object is created and the received packet along with the additional parameters is sene to the other user's websocket (if present)

```
packet['created'] = msg.get_formatted_create_datetime()
packet['sender_name'] = msg.sender.username
packet['message_id'] = msg.id
```

new-user Sends connected client list of currently active users.

```
# Get list list of current active users
users = [
    {'username': username, 'uuid': uuid_str}
    for username, uuid_str in ws_connections.values()
]

# Make packet with list of new users (sorted by username)
packet = {
    'type': 'users-changed',
    'value': sorted(users, key=lambda i: i['username'])
}
```

online Informs the users when someone of other has gone online.

{'type': 'gone-online', 'usernames': [user_owner.username]}

offline Distributes the users 'gone offline' status to everyone he has dialog with {'type': 'gone-offline', 'username': user_owner.username}

check-online Same as online, except that it is used to provide the user that has gone online with information about who of his dialogs' users is online.

is-typing Shows message to opponent if the user is typing a message

{'type': 'opponent-typing', 'username': user_opponent}

read_message Send message to user if the opponent has read the message Also sets the message.read to *True*.

{'type': 'opponent-read-message', 'username': user_opponent, 'message_id': message_id}

CHAPTER

FIVE

SETTINGS

You should specify settings in your settings.py like this:

```
CHAT_WS_SERVER_HOST = 'localhost'
CHAT_WS_SERVER_PORT = 5002
CHAT_WS_SERVER_PROTOCOL = 'ws'
DATETIME_FORMAT = "d.m.Y H:i:s"
```

Here's a list of available settings:

```
CHAT_WS_SERVER_PROTOCOL - 'ws' or 'wss'
CHAT_WS_SERVER_HOST - 'localhost' or ip or domain
CHAT_WS_SERVER_PORT - websocket application port
DATETIME_FORMAT - "d.m.Y H:i:s" - format for datetimes
```

CHAPTER

SIX

ADMIN

Application provides django admin intergration for `Dialog` and `Message` models.

In order to provide custom admin representation, first you have to unregister existing:

```
from django_private_chat.models import Dialog, Message
admin.site.unregister(Dialog)
admin.site.unregister(Message)

// your example admin
class DialogAdmin(admin.ModelAdmin):
    list_display = ('id',)
admin.site.register(Dialog, DialogAdmin)
```

CHAPTER

SEVEN

STARTING THE SERVER

Application provides two managements commands, `run_chat_server` and `run_chat_server_uvloop`.

That means that asyncio server is started SEPARATELY from the main Django application. You can also supply optional "path/to/cert.pem" to the command to use wss.

What management command do is they simply get the asyncio/uvloop event loop, add handlers for different message types to it and run the loop forever.

CHAPTER

EIGHT

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:

8.1 Types of Contributions

8.1.1 Report Bugs

Report bugs at https://github.com/Bearle/django-private-chat/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.

8.1.2 Fix Bugs

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

8.1.3 Implement Features

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

8.1.4 Write Documentation

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

8.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/Bearle/django-private-chat/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 :)

8.2 Get Started!

Ready to contribute? Here's how to set up *django-private-chat* for local development.

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

```
$ git clone git@github.com:your_name_here/django-private-chat.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-private-chat
$ cd django-private-chat/
$ 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 django_private_chat tests
$ python setup.py test
$ tox
```

   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.

8.3 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.

8.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_django_private_chat
```

CHAPTER

NINE

CREDITS

9.1 Development Lead

- delneg <tech@bearle.ru>
- guitarmustafa <oleg5432101@rambler.ru>

9.2 Contributors

None yet. Why not be the first?

History

CHAPTER

TEN

0.3.0 (2020-05-03)

- Update deps, example to Django 2.2.12
- Move to async/await syntax

CHAPTER

ELEVEN

0.2.2 (2018-12-12)

- Fix read_message_handler by idonoso

CHAPTER

TWELVE

0.2.1 (2018-12-07)

- Compatibility with python3.7 by Emeka Icha

CHAPTER

THIRTEEN

0.2.0 (2018-10-22)

- Added WSS fix for python 3.4 & 3.5

CHAPTER

FOURTEEN

0.1.9 (2018-07-16)

- Added WSS support by @zsmith3

CHAPTER

FIFTEEN

0.1.8 (2018-03-23)

- Fixed time in Message model to be timezone-aware

CHAPTER

SIXTEEN

0.1.7 (2018-03-20)

- Additions for django 2.0

CHAPTER

SEVENTEEN

0.1.6 (2017-04-11)

- Fixed bugs with static files and added comment about extra_js block to readme

CHAPTER

EIGHTEEN

0.1.5 (2017-03-11)

- Added flashing other user button when he sent you a message and you're in another dialog

CHAPTER

NINETEEN

0.1.4 (2017-02-12)

- Added support for django 1.8,1.9

CHAPTER

TWENTY

0.1.3 (2017-02-11)

- Removed uvloop from requirements

CHAPTER

TWENTYONE

0.1.2 (2017-02-11)

- Fixed i18n not loaded in dialogs template bug

CHAPTER

TWENTYTWO

0.1.1 (2017-02-10)

- Added migrations.

CHAPTER

TWENTYTHREE

0.1.0 (2017-02-10)

- First release on PyPI.