dasf-broker-django

Philipp S. Sommer

Feb 16, 2023

CONTENTS:

A Django-based message broker for the Data Analytics Software Framework (DASF)

Warning: This package is work in progress, especially it's documentation. Stay tuned for updates and discuss with us at https://gitlab.hzdr.de/hcdc/django/dasf-broker-django

CHAPTER

ONE

INSTALLATION

To install the *dasf-broker-django* package for your Django project, you need to follow two steps:

1. *Install the package*
2. *Add the app to your Django project*

1.1 Installation from PyPi

The recommended way to install this package is via pip and PyPi via:

```
pip install dasf-broker-django
```

Or install it directly from the source code repository on Gitlab via:

```
pip install git+https://gitlab.hzdr.de/hcdc/django/dasf-broker-django.git
```

The latter should however only be done if you want to access the development versions.

1.2 Install the Django App for your project

To use the *dasf-broker-django* package in your Django project, you need to add the app to your *INSTALLED_APPS*, configure your *urls.py*, run the migration, add a login button in your templates. Here are the step-by-step instructions:

1. Add the *dasf_broker* app to your *INSTALLED_APPS*
2. in your projects urlconf (see **:setting:`ROOT_URLCONF`**), add include *dasf_broker.urls* via:

   ```
   from django.urls import include, path

   urlpatterns += [
       path("dasf-broker-django/", include("dasf_broker.urls")),
    ]
   ```

3. Run `python manage.py migrate` to add models to your database
4. Configure the app to your needs (see *Configuration options*).

That's it! For further adaption to you Django project, please head over to the *Configuration options*. You can also have a look into the `testproject` in the source code repository for a possible implementation.

1.3 Installation for development

Please head over to our *contributing guide* for installation instruction for development.

CHAPTER

TWO

CONFIGURATION OPTIONS

2.1 Configuration settings

The following settings have an effect on the app.

DASF_CREATE_TOPIC_ON_MESSAGE	Create new topics on message
DASF_STORE_MESSAGES	Shall the messages be stored?
DASF_STORE_RESPONSE_MESSAGES	Shall the messages to response topics be stored?
DASF_STORE_SOURCE_MESSAGES	Shall the source messages be stored?
DASF_WEBSOCKET_URL_ROUTE	URL route for the websocket
ROOT_URL	root URL to your application

StoreMessageOptions(value)	An enumeration.

CHAPTER

THREE

API REFERENCE

3.1 App settings

This module defines the settings options for the `dasf-broker-django` app.

Data:

DASF_CREATE_TOPIC_ON_MESSAGE	Create new topics on message
DASF_STORE_MESSAGES	Shall the messages be stored?
DASF_STORE_RESPONSE_MESSAGES	Shall the messages to response topics be stored?
DASF_STORE_SOURCE_MESSAGES	Shall the source messages be stored?
DASF_WEBSOCKET_URL_ROUTE	URL route for the websocket
ROOT_URL	root URL to your application

Classes:

StoreMessageOptions(value)	An enumeration.

dasf_broker.app_settings.**DASF_CREATE_TOPIC_ON_MESSAGE**: bool = **True**

Create new topics on message

This flag controls if new topics are created when a message comes from a producer. If False, messages with non-existing topics are ignored.

Note that a user also needs the *dasf_broker.add_BrokerTopic* permission to create topics.

dasf_broker.app_settings.**DASF_STORE_MESSAGES**: *StoreMessageOptions* = **StoreMessageOptions.CACHE**

Shall the messages be stored?

This flag controls whether the message broker caches messages from producers until they are consumed by the consumer. This is useful if the consumer looses connection to the server. This settings can take three different values:

"disabled"
: The message is not stored at all

"cache"
: The message is stored and removed ones one of the potential *consumers* acknowledges the message

"cacheall"
: The message is stored and removed ones **all** *consumers* acknowledged the message

"store"
The message and response topics are stored forever and are not automatically removed

See also:

DASF_STORE_SOURCE_MESSAGES, *DASF_STORE_RESPONSE_MESSAGES*

dasf_broker.app_settings.**DASF_STORE_RESPONSE_MESSAGES: *StoreMessageOptions* = StoreMessageOptions.CACHE**

Shall the messages to response topics be stored?

This flag controls whether the message broker caches messages from producers to topics that are marked as response topic. If this setting is not set, we use the DASF_STORE_MESSAGES setting.

dasf_broker.app_settings.**DASF_STORE_SOURCE_MESSAGES: *StoreMessageOptions* = StoreMessageOptions.CACHE**

Shall the source messages be stored?

This flag controls whether the message broker caches messages from producers to topics that are **not** marked as response topic. If this setting is not set, we use the DASF_STORE_MESSAGES setting.

dasf_broker.app_settings.**DASF_WEBSOCKET_URL_ROUTE: str = 'ws/'**

URL route for the websocket

This setting controls, where we expect to find the websockets. As there is no analog to `django.urls.reverse()` for channels, you should use this setting in your `asgi.py` file to include the routes of this package.

Examples

In your `asgi.py` file, include it like:

```
from channels.routing import ProtocolTypeRouter, URLRouter
from django.core.asgi import get_asgi_application
from channels.auth import AuthMiddlewareStack
import dasf_broker.routing as dasf_routing
from dasf_broker.app_settings import DASF_WEBSOCKET_URL_ROUTE

application = ProtocolTypeRouter(
    {
        "http": get_asgi_application(),
        "websocket": AuthMiddlewareStack(
            URLRouter(
                [
                    path(
                        DASF_WEBSOCKET_URL_ROUTE,
                        URLRouter(dasf_routing.websocket_urlpatterns),
                    )
                ]
            )
        ),
    }
)
```

dasf_broker.app_settings.**ROOT_URL: str | None = None**

root URL to your application

You can use this setting if you are behind a reverse proxy and the host names, etc. are not handled correctly.

If you leave this empty, we will use the `build_absolute_uri` method of the http request.

Examples

A standard value for this would be `http://localhost:8000`

class dasf_broker.app_settings.**StoreMessageOptions**(*value*)

Bases: `str`, `Enum`

An enumeration.

Attributes:

CACHE	
CACHEALL	
DISABLED	
STORE	

CACHE = 'cache'

CACHEALL = 'cacheall'

DISABLED = 'disabled'

STORE = 'store'

3.2 URL config

URL patterns of the dasf-broker-django to be included via:

```
from django.urls import include, path

urlpatters = [
    path(
        "dasf-broker-django",
        include("dasf_broker.urls"),
    ),
]
```

Data:

app_name	App name for the dasf-broker-django to be used in calls to `django.urls.reverse()`
urlpatterns	urlpattern for the Helmholtz AAI

dasf_broker.urls.**app_name = 'dasf_broker'**

App name for the dasf-broker-django to be used in calls to `django.urls.reverse()`

dasf_broker.urls.**urlpatterns**: list[Any] = [<URLPattern '<slug>/status/' [name='brokertopic-status']>, <URLPattern '<slug>/ping/' [name='brokertopic-ping']>]

urlpattern for the Helmholtz AAI

3.3 Models

Models for the dasf-broker-django app.

Models:

BrokerMessage(*args, **kwargs)	A message sent to the broker.
BrokerTopic(*args, **kwargs)	A topic for producing and consuming requests via websocket
ResponseTopic(*args, **kwargs)	A topic that accepts responses for messages.

Classes:

BrokerTopicManager(*args, **kwargs)	A manager for broker topics.
BrokerTopicQuerySet([model, query, using, hints])	A queryset for broker topics.

class dasf_broker.models.**BrokerMessage**(**args*, ***kwargs*)

Bases: Model

A message sent to the broker.

Miscellaneous:

DoesNotExist
MultipleObjectsReturned

Model Fields:

content	A wrapper for a deferred-loading field.
context	A wrapper for a deferred-loading field.
date_created	A wrapper for a deferred-loading field.
id	A wrapper for a deferred-loading field.
message_id	A wrapper for a deferred-loading field.
topic	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.
user	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

Attributes:

delivered_to	Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.
delivered_to_all	Test if the message has been delivered to all consumers.
is_response	Is this message a response to a DASF request?
objects	
responsetopic_set	Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.
topic_id	
user_id	

Methods:

get_next_by_date_created(*[, field, is_next])	
get_previous_by_date_created(*[, field, is_next])	
send()	Send the message via the websocket.

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

content

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

context

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

date_created

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

delivered_to

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

property delivered_to_all: bool

Test if the message has been delivered to all consumers.

get_next_by_date_created(*, *field=<django.db.models.fields.DateTimeField: date_created>*, *is_next=True*, ***kwargs*)

get_previous_by_date_created(*, *field=<django.db.models.fields.DateTimeField: date_created>*, *is_next=False*, ***kwargs*)

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property is_response: bool

Is this message a response to a DASF request?

message_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

responsetopic_set

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

send()

Send the message via the websocket.

topic

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

topic_id

user

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

user_id

class dasf_broker.models.**BrokerTopic**(**args*, ***kwargs*)

Bases: Model

A topic for producing and consuming requests via websocket

Miscellaneous:

DoesNotExist	
MultipleObjectsReturned	

Classes:

StoreMessageChoices(value)	Choices for storing messages.

Attributes:

availability	Get the online/offline status for the topic.
brokermessage_set	Accessor to the related objects manager on the reverse side of a many-to-one relation.
consumers	
effective_store_messages	Get the store message rule for this topic.
is_response_topic	Is this topic a responsetopic?
objects	
producers	
responsetopic	Accessor to the related object on the reverse side of a one-to-one relation.
responsetopics	Accessor to the related objects manager on the reverse side of a many-to-one relation.
status_viewers	

Methods:

build_websocket_url(request[, route])	
create_and_send_message(user, content)	Create and send a message for the user
get_next_by_date_created(*[, field, is_next])	
get_outstanding_messages([user])	Get the messages that still need to be send.
get_previous_by_date_created(*[, field, is_next])	
get_store_messages_display(*[, field])	
get_websocket_url(request)	Get the websocket url for this topic.
ping()	Create a ping message and send it to the consumer.

Model Fields:

date_created	A wrapper for a deferred-loading field.
garbage_collect_on	A wrapper for a deferred-loading field.
id	A wrapper for a deferred-loading field.
is_public	A wrapper for a deferred-loading field.
last_ping	A wrapper for a deferred-loading field.
last_pong	A wrapper for a deferred-loading field.
slug	A wrapper for a deferred-loading field.
store_messages	A wrapper for a deferred-loading field.
supports_dasf	A wrapper for a deferred-loading field.

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

class StoreMessageChoices(*value*)

Bases: TextChoices

Choices for storing messages.

Attributes:

CACHE	
CACHEALL	
DISABLED	
STORE	

CACHE = 'cache'

CACHEALL = 'cacheall'

DISABLED = 'disabled'

`STORE = 'store'`

availability

Get the online/offline status for the topic.

This value can be `True`, `False` or `None`:

None
The status is unknown. This occurs when the last ping was more than two minutes ago or the topic has never, been pinged.

False
The was no pong yet or the last pong was before the last ping and the last ping was less than two minutes ago.

True
The topic is online, i.e. we received a pong after the last ping and the last ping was less then two minutes ago.

brokermessage_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

classmethod build_websocket_url(*request*, *route: str | None = None*) → str

property consumers: models.QuerySet[User]

create_and_send_message(*user: User*, *content: Dict*)

Create and send a message for the user

date_created

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property effective_store_messages: *StoreMessageChoices*

Get the store message rule for this topic.

garbage_collect_on

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_next_by_date_created(*, *field=<django.db.models.fields.DateTimeField: date_created>*, *is_next=True*, ***kwargs*)

get_outstanding_messages(*user: User | None = None*) → models.QuerySet[*BrokerMessage*]

Get the messages that still need to be send.

Parameters
user (*Optional[User]*) – The user for whom to send the messages. If None, the messages will be returned that have not yet been acknowledged at all.

Returns
A QuerySet of messages

Return type
models.QuerySet[*BrokerMessage*]

get_previous_by_date_created(*, *field=<django.db.models.fields.DateTimeField: date_created>*, *is_next=False*, ***kwargs*)

get_store_messages_display(*, *field=<django.db.models.fields.CharField: store_messages>*)

get_websocket_url(*request*) → str

Get the websocket url for this topic.

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

is_public

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property is_response_topic: bool

Is this topic a responsetopic?

last_ping

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

last_pong

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <dasf_broker.models.BrokerTopicManager object>

ping()

Create a ping message and send it to the consumer.

property producers: models.QuerySet[User]

responsetopic

Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

responsetopics

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

slug

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property status_viewers: models.QuerySet[User]

store_messages

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

supports_dasf

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class dasf_broker.models.**BrokerTopicManager**(**args*, ***kwargs*)

Bases: `ManagerFromBrokerTopicQuerySet`

A manager for broker topics.

class dasf_broker.models.**BrokerTopicQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

Bases: `QuerySet`

A queryset for broker topics.

Methods:

filter_offline(*args)	Query all online broker topics.
filter_online(*args, **kwargs)	Query all online broker topics.
filter_unknown_availability(*args)	Query all topics where the availability is unknown.

filter_offline(**args*)

Query all online broker topics.

filter_online(**args*, ***kwargs*)

Query all online broker topics.

filter_unknown_availability(**args*)

Query all topics where the availability is unknown.

class dasf_broker.models.**ResponseTopic**(**args*, ***kwargs*)

Bases: *BrokerTopic*

A topic that accepts responses for messages.

Miscellaneous:

DoesNotExist
MultipleObjectsReturned

Model Fields:

brokertopic_ptr	Accessor to the related object on the forward side of a one-to-one relation.
is_response_for	Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

Attributes:

brokertopic_ptr_id	
is_response_for_id	
is_response_topic	Is this topic a responsetopic?
source_messages	Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

exception DoesNotExist

Bases: *DoesNotExist*

exception MultipleObjectsReturned

Bases: *MultipleObjectsReturned*

brokertopic_ptr: *BrokerTopic*

Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

brokertopic_ptr_id

is_response_for

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

is_response_for_id

property is_response_topic: bool

Is this topic a responsetopic?

source_messages

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

3.4 Views

Views of the dasf-broker-django app to be imported via the url config (see *dasf_broker.urls*).

Classes:

BrokerTopicPingView(**kwargs)	View to ping a broker topic.
BrokerTopicStatusView(**kwargs)	Get a hint on the status of a broker topic.
HttpResponseServiceUnavailable([content])	

class dasf_broker.views.**BrokerTopicPingView**(***kwargs*)

Bases: `PermissionRequiredMixin`, `SingleObjectMixin`, `View`

View to ping a broker topic.

Attributes:

accept_global_perms
permission_required

Methods:

check_permissions(request)	Checks if *request.user* has all permissions returned by *get_required_permissions* method.
get(request, *args, **kwargs)	
post(request, *args, **kwargs)	

Models:

model	alias of *BrokerTopic*

accept_global_perms = True

check_permissions(*request*)

Checks if *request.user* has all permissions returned by *get_required_permissions* method.

Parameters
request – Original request.

get(*request*, **args*, ***kwargs*)

model

alias of *BrokerTopic* **Miscellaneous:**

DoesNotExist	
MultipleObjectsReturned	

Classes:

StoreMessageChoices(value)	Choices for storing messages.

Attributes:

availability	Get the online/offline status for the topic.
brokermessage_set	Accessor to the related objects manager on the reverse side of a many-to-one relation.
consumers	
effective_store_messages	Get the store message rule for this topic.
is_response_topic	Is this topic a responsetopic?
objects	
producers	
responsetopic	Accessor to the related object on the reverse side of a one-to-one relation.
responsetopics	Accessor to the related objects manager on the reverse side of a many-to-one relation.
status_viewers	

Methods:

build_websocket_url(request[, route])	
create_and_send_message(user, content)	Create and send a message for the user
get_next_by_date_created(*[, field, is_next])	
get_outstanding_messages([user])	Get the messages that still need to be send.
get_previous_by_date_created(*[, field, is_next])	
get_store_messages_display(*[, field])	
get_websocket_url(request)	Get the websocket url for this topic.
ping()	Create a ping message and send it to the consumer.

Model Fields:

date_created	A wrapper for a deferred-loading field.
garbage_collect_on	A wrapper for a deferred-loading field.
id	A wrapper for a deferred-loading field.
is_public	A wrapper for a deferred-loading field.
last_ping	A wrapper for a deferred-loading field.
last_pong	A wrapper for a deferred-loading field.
slug	A wrapper for a deferred-loading field.
store_messages	A wrapper for a deferred-loading field.
supports_dasf	A wrapper for a deferred-loading field.

permission_required = 'dasf_broker.can_produce'

post(*request*, **args*, ***kwargs*)

class dasf_broker.views.**BrokerTopicStatusView**(***kwargs*)

Bases: PermissionRequiredMixin, BaseDetailView

Get a hint on the status of a broker topic.

Attributes:

accept_global_perms
any_perm
permission_required

Methods:

check_permissions(request)	Checks if *request.user* has all permissions returned by *get_required_permissions* method.
render_to_response(context)	

Models:

model	alias of *BrokerTopic*

accept_global_perms = True

any_perm = True

check_permissions(*request*)

Checks if *request.user* has all permissions returned by *get_required_permissions* method.

Parameters
request – Original request.

`model`

alias of *BrokerTopic* **Miscellaneous:**

`DoesNotExist`	
`MultipleObjectsReturned`	

Classes:

`StoreMessageChoices`(value)	Choices for storing messages.

Attributes:

`availability`	Get the online/offline status for the topic.
`brokermessage_set`	Accessor to the related objects manager on the reverse side of a many-to-one relation.
`consumers`	
`effective_store_messages`	Get the store message rule for this topic.
`is_response_topic`	Is this topic a responsetopic?
`objects`	
`producers`	
`responsetopic`	Accessor to the related object on the reverse side of a one-to-one relation.
`responsetopics`	Accessor to the related objects manager on the reverse side of a many-to-one relation.
`status_viewers`	

Methods:

`build_websocket_url`(request[, route])	
`create_and_send_message`(user, content)	Create and send a message for the user
`get_next_by_date_created`(*[, field, is_next])	
`get_outstanding_messages`([user])	Get the messages that still need to be send.
`get_previous_by_date_created`(*[, field, is_next])	
`get_store_messages_display`(*[, field])	
`get_websocket_url`(request)	Get the websocket url for this topic.
`ping`()	Create a ping message and send it to the consumer.

Model Fields:

date_created	A wrapper for a deferred-loading field.
garbage_collect_on	A wrapper for a deferred-loading field.
id	A wrapper for a deferred-loading field.
is_public	A wrapper for a deferred-loading field.
last_ping	A wrapper for a deferred-loading field.
last_pong	A wrapper for a deferred-loading field.
slug	A wrapper for a deferred-loading field.
store_messages	A wrapper for a deferred-loading field.
supports_dasf	A wrapper for a deferred-loading field.

permission_required = ['dasf_broker.can_view_status', 'dasf_broker.can_produce', 'dasf_broker.can_consume']

render_to_response(*context*)

class dasf_broker.views.**HttpResponseServiceUnavailable**(*content=b''*, **args*, ***kwargs*)

Bases: HttpResponse

Attributes:

status_code

status_code = 503

CHAPTER

FOUR

CONTRIBUTION AND DEVELOPMENT HINTS

The dasf-broker-django project is developed by the Helmholtz-Zentrum Hereon. It is open-source as we believe that this package can be helpful for multiple other django applications, and we are looking forward for your feedback, questions and especially for your contributions.

- If you want to ask a question, are missing a feature or have comments on the docs, please open an issue at the source code repository
- If you have suggestions for improvement, please let us know in an issue, or fork the repository and create a merge request. See also *Contributing in the development*.

4.1 Contributing in the development

Thanks for your wish to contribute to this app!! The source code of the dasf-broker-django package is hosted at https://gitlab.hzdr.de/hcdc/django/dasf-broker-django. It's an open gitlab where you can register via GitHub, or via the Helmholtz AAI. Once you created an account, you can fork this repository to your own user account and implement the changes. Afterwards, please make a merge request into the main repository. If you have any questions, please do not hesitate to create an issue on gitlab and contact the developers.

Warning: For local development, you need a redis server available. They can be configured via environment variables (`REDIS_HOST`, see the `settings.py` file in the testproject).

Once you created you fork, you can clone it via

```
git clone https://gitlab.hzdr.de/<your-user>/dasf-broker-django.git
```

and install it in development mode with the *[dev]* option via:

```
pip install -e ./dasf-broker-django/[dev]
```

Once you installed the package, run the migrations:

```
cd dasf-broker-django/
python manage.py migrate
```

which will setup the database for you.

4.1.1 Fixing the docs

The documentation for this package is written in restructured Text and built with sphinx and deployed on readthedocs.

If you found something in the docs that you want to fix, head over to the `docs` folder and build the docs with *make html* (or *make.bat* on windows). The docs are then available in `docs/_build/html/index.html` that you can open with your local browser.

Implement your fixes in the corresponding `.rst`-file and push them to your fork on gitlab.

4.1.2 Contributing to the code

We use automated formatters (see their config in `pyproject.toml` and `setup.cfg`), namely

- Black for standardized code formatting
- blackdoc for standardized code formatting in documentation
- Flake8 for general code quality
- isort for standardized order in imports.
- mypy for static type checking on type hints

We highly recommend that you setup pre-commit hooks to automatically run all the above tools every time you make a git commit. This can be done by running:

```
pre-commit install
```

from the root of the repository. You can skip the pre-commit checks with `git commit --no-verify` but note that the CI will fail if it encounters any formatting errors.

You can also run the pre-commit step manually by invoking:

```
pre-commit run --all-files
```

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX

R

S

T

U