
PyFrost

Release 0.2.1

Dennis Pham

Apr 17, 2020

CONTENTS

1	The Official Client	3
2	The Source Code	5
3	Contents	7
3.1	frost package	7
3.1.1	Subpackages	7
3.1.1.1	frost.client package	7
3.1.1.2	frost.ext package	20
3.1.1.3	frost.server package	21
3.1.2	Module contents	35
	Python Module Index	39
	Index	41

PyFrost

THE OFFICIAL CLIENT

The official client for this library is under development. You can test it out and/or contribute to the [repository](#).

THE SOURCE CODE

The source is available [here](#) on GitHub. Feel free to contribute to this project.

CONTENTS

3.1 frost package

3.1.1 Subpackages

3.1.1.1 frost.client package

Subpackages

frost.client.events package

Submodules

frost.client.events.cogs module

class frost.client.events.cogs.Auth

Bases: *frost.ext.cog.Cog*

Deals with user authentication. route='authentication'

post_login() → None

Deals with the response received from the server after a login attempt.

Parameters **data** (*Dict[str, Any]*) – Data received from the server

post_register() → None

Deals with the response received from the server after a registration attempt.

Parameters **data** (*Dict[str, Any]*) – Data received from the server

route = 'authentication'

class frost.client.events.cogs.Msgs

Bases: *frost.ext.cog.Cog*

Deals with user messages. route='messages'

new() → None

Deals with new user messages and stores them in *frost.client.events.events.Messages*.

Parameters **data** (*Dict[str, Any]*) – Data received from the server

post_new() → None

Stores the response status of a send message attempt.

Parameters **data** (*Dict[str, Any]*) – Data received from the server

post_room() → None
Deals with the post response from requesting all messages in a specific room. Messages are stored in *frost.client.events.events.Messages*.

Parameters **data** (*Dict[str, Any]*) – Data received from the server

route = 'messages'

class *frost.client.events.cogs.Rooms*
Bases: *frost.ext.cog.Cog*

Deals with room within a server. **route**='rooms'

new_room_member() → None
Adds a member to a specific room when they join.

Parameters **data** (*Dict[str, Any]*) – Data received from the server

post_all_joined() → None
Deals with the post response from a request for all joined rooms.

Parameters **data** (*Dict[str, Any]*) – Data received from the server

post_create() → None
Deals with the post response from a room creation request.

Parameters **data** (*Dict[str, Any]*) – Data received from the server

post_invite_code() → None
Deals with the post response from a room invite code request.

Parameters **data** (*Dict[str, Any]*) – Data received from the server

post_join() → None
Deals with the post response from a join room request.

Parameters **data** (*Dict[str, Any]*) – Data received from the server

post_leave() → None
Deals with the post response from a leave room request.

Parameters **data** (*Dict[str, Any]*) – Data received from the server

post_members() → None
Deals with the post response from a request for all members in a room.

Parameters **data** (*Dict[str, Any]*) – Data received from the server

remove_room_member() → None
Removes a member from a specific room when they leave.

Parameters **data** (*Dict[str, Any]*) – Data received from the server

route = 'rooms'

frost.client.events.events module**class** frost.client.events.events.**EventStatus**

Bases: object

Stores the current status of all events.

create_room = None**get_invite_code** = None**get_joined_rooms** = None**get_room_members** = None**get_room_msgs** = None**classmethod** **get_status** (*item: str*) → Optional[int]

Returns the current status of the specified item and resets it to None after.

Returns The current event status**Return type** Optional[int]**join_room** = None**leave_room** = None**login** = None**register** = None**send_msg** = None**class** frost.client.events.events.**Messages**

Bases: object

All messages will be stored in this class.

classmethod **add_new_msgs** (*msgs: Dict[int, Dict[str, Dict[str, Union[str, Dict[str, str]]]]]*) →None
Save new messages, grouped by room.**Parameters** **msgs** (*Dict[int, Dict[str, Dict[str, Union[str, Dict[str, str]]]]]*) – The messages to save**all** = {}

All messages stored.

classmethod **clear** () → NoneClears `cls.all` and `cls.__new`.**classmethod** **get_new_msgs** () → Dict[int, Dict[str, Dict[str, Union[str, Dict[str, str]]]]]

Returns new, unread messages.

Returns The new messages**Return type** Dict[int, Dict[str, Dict[str, Union[str, Dict[str, str]]]]]

Module contents

class `frost.client.events.Auth`

Bases: `frost.ext.cog.Cog`

Deals with user authentication. `route='authentication'`

post_login() → None

Deals with the response received from the server after a login attempt.

Parameters **data** (`Dict[str, Any]`) – Data received from the server

post_register() → None

Deals with the response received from the server after a registration attempt.

Parameters **data** (`Dict[str, Any]`) – Data received from the server

route = `'authentication'`

class `frost.client.events.Msgs`

Bases: `frost.ext.cog.Cog`

Deals with user messages. `route='messages'`

new() → None

Deals with new user messages and stores them in `frost.client.events.events.Messages`.

Parameters **data** (`Dict[str, Any]`) – Data received from the server

post_new() → None

Stores the response status of a send message attempt.

Parameters **data** (`Dict[str, Any]`) – Data received from the server

post_room() → None

Deals with the post response from requesting all messages in a specific room. Messages are stored in `frost.client.events.events.Messages`.

Parameters **data** (`Dict[str, Any]`) – Data received from the server

route = `'messages'`

class `frost.client.events.Rooms`

Bases: `frost.ext.cog.Cog`

Deals with room within a server. `route='rooms'`

new_room_member() → None

Adds a member to a specific room when they join.

Parameters **data** (`Dict[str, Any]`) – Data received from the server

post_all_joined() → None

Deals with the post response from a request for all joined rooms.

Parameters **data** (`Dict[str, Any]`) – Data received from the server

post_create() → None

Deals with the post response from a room creation request.

Parameters **data** (`Dict[str, Any]`) – Data received from the server

post_invite_code() → None

Deals with the post response from a room invite code request.

Parameters **data** (`Dict[str, Any]`) – Data received from the server

```

post_join() → None
    Deals with the post response from a join room request.

    Parameters data (Dict[str, Any]) – Data received from the server

post_leave() → None
    Deals with the post response from a leave room request.

    Parameters data (Dict[str, Any]) – Data received from the server

post_members() → None
    Deals with the post response from a request for all members in a room.

    Parameters data (Dict[str, Any]) – Data received from the server

remove_room_member() → None
    Removes a member from a specific room when they leave.

    Parameters data (Dict[str, Any]) – Data received from the server

route = 'rooms'

class frost.client.events.Messages
    Bases: object

    All messages will be stored in this class.

    classmethod add_new_msgs (msgs: Dict[int, Dict[str, Dict[str, Union[str, Dict[str, str]]]]]) →
        None
        Save new messages, grouped by room.

        Parameters msgs (Dict[int, Dict[str, Dict[str, Union[str, Dict[str, str]]]]]) – The messages to save

    all = {}
        All messages stored.

    classmethod clear() → None
        Clears cls.all and cls.__new.

    classmethod get_new_msgs() → Dict[int, Dict[str, Dict[str, Union[str, Dict[str, str]]]]]
        Returns new, unread messages.

        Returns The new messages

        Return type Dict[int, Dict[str, Dict[str, Union[str, Dict[str, str]]]]]

class frost.client.events.EventStatus
    Bases: object

    Stores the current status of all events.

    create_room = None

    get_invite_code = None

    get_joined_rooms = None

    get_room_members = None

    get_room_msgs = None

    classmethod get_status (item: str) → Optional[int]
        Returns the current status of the specified item and resets it to None after.

        Returns The current event status

        Return type Optional[int]

```

```
join_room = None
leave_room = None
login = None
register = None
send_msg = None
```

frost.client.socketio package

Submodules

frost.client.socketio.base_client module

```
class frost.client.socketio.base_client.BaseClient (ip: str = '127.0.0.1', port: int = 5555)
```

Bases: object

The base client to connect to a server & send and receive data.

Parameters

- **ip** (*str, optional*) – The IP address of the server to connect to, defaults to ‘127.0.0.1’
- **port** (*int, optional*) – The port of the server to connect to, defaults to 5555

close () → None

Close the connection to the server.

connect () → None

Connect and establish a connect to the server.

recieve () → Any

Recieve data from the server.

Returns Data received from the server

Return type Any

send (*data: Any*) → None

Send data to the server.

Parameters **data** (*Any*) – Data to send to the server

frost.client.socketio.utils module

```
class frost.client.socketio.utils.threaded (*args: Any, **kwargs: Any)
```

Bases: object

A decorator to thread a function/method.

Module contents

class `frost.client.socketio.BaseClient` (*ip*: *str* = '127.0.0.1', *port*: *int* = 5555)

Bases: `object`

The base client to connect to a server & send and receive data.

Parameters

- **ip** (*str*, *optional*) – The IP address of the server to connect to, defaults to '127.0.0.1'
- **port** (*int*, *optional*) – The port of the server to connect to, defaults to 5555

close () → `None`

Close the connection to the server.

connect () → `None`

Connect and establish a connect to the server.

recieve () → `Any`

Recieve data from the server.

Returns Data received from the server

Return type `Any`

send (*data*: `Any`) → `None`

Send data to the server.

Parameters **data** (`Any`) – Data to send to the server

class `frost.client.socketio.threaded` (**args*: `Any`, ***kwargs*: `Any`)

Bases: `object`

A decorator to thread a function/method.

Submodules

frost.client.auth module

`frost.client.auth.get_auth` (*func*: `Callable`) → `Callable`

A decorator to get the saved auth token and id.

Parameters **func** (`Callable`) – The function that is being wrapped

Returns The inner execute function

Return type `Callable`

`frost.client.auth.get_auth_token`

Returns the saved auth token.

Returns The auth token saved in `.frost`

Return type `Optional[str]`

`frost.client.auth.get_id`

Returns the saved id.

Returns The ID saved in `.frost`

Return type `Optional[str]`

frost.client.client module**class** `frost.client.client.FrostClient` (*ip*: *str* = '127.0.0.1', *port*: *int* = 5555)Bases: `frost.client.socketio.base_client.BaseClient`

The Frost Client.

Parameters

- **ip** (*str*, *optional*) – The IP address of the server to connect to, defaults to '127.0.0.1'
- **port** (*int*, *optional*) – The port of the server to connect to, defaults to 5555

connect () → None

Connect to the server and begin listening for events.

create_room (*room_name*: *str*, *token*: *str*, *id_*: *str*) → None

Create a new room in a server.

Parameters

- **room_name** (*str*) – The name of the room to create
- **token** (*str*) – The user's token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user's ID, auto filled by `frost.client.auth.get_auth()`

get_invite_code (*room_id*: *int*, *token*: *str*, *id_*: *str*) → None

Get the invite code of a room in a server.

Parameters

- **room_id** (*int*) – The ID of the room to get an invite code from
- **token** (*str*) – The user's token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user's ID, auto filled by `frost.client.auth.get_auth()`

get_joined_rooms (*token*: *str*, *id_*: *str*) → None

Get all the joined rooms of the currently logged in user.

Parameters

- **token** (*str*) – The user's token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user's ID, auto filled by `frost.client.auth.get_auth()`

get_room_members (*room_id*: *int*, *token*: *str*, *id_*: *str*) → None

Get all the members of a specific room.

Parameters

- **room_id** (*int*) – The ID of the room to get the members of
- **token** (*str*) – The user's token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user's ID, auto filled by `frost.client.auth.get_auth()`

get_room_msgs (*room_id*: *int*, *token*: *str*, *id_*: *str*) → None

Get all messages from a specific room in a server.

Parameters

- **room_id** (*int*) – The ID of the room to get the messages from
- **token** (*str*) – The user's token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user's ID, auto filled by `frost.client.auth.get_auth()`

join_room (*invite_code: str, token: str, id_: str*) → None

Join a room in a server with an invite code.

Parameters

- **invite_code** (*str*) – The invite code the room to join
- **token** (*str*) – The user's token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user's ID, auto filled by `frost.client.auth.get_auth()`

leave_room (*room_id: int, token: str, id_: str*) → None

Leave a joined room in a server.

Parameters

- **room_id** (*int*) – The ID of the room to leave
- **token** (*str*) – The user's token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user's ID, auto filled by `frost.client.auth.get_auth()`

login (*username: str, password: str*) → None

Login to the server.

Parameters

- **username** (*str*) – The username of the account
- **password** (*str*) – The password of the account

register (*username: str, password: str*) → None

Register an account on the server.

Parameters

- **username** (*str*) – The desired username of the account
- **password** (*str*) – The desired password of the account

send_msg (*room_id: int, msg: str, token: str, id_: str*) → None

Send a message to other users on a server in a specific room.

Parameters

- **room_id** (*int*) – The ID of the room to send the message to
- **msg** (*str*) – The desired message to send
- **token** (*str*) – The user's token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user's ID, auto filled by `frost.client.auth.get_auth()`

frost.client.headers module

class `frost.client.headers.Status`

Bases: `enum.Enum`

The status enums

DUPLICATE_ROOM_NAME: `int = 6`

DUPLICATE_USERNAME: `int = 3`

EMPTY_ROOM_NAME: `int = 5`

INVALID_AUTH: `int = 1`

```
INVALID_INVITE: int = 7
PERMISSION_DENIED: int = 2
ROOM_NOT_FOUND: int = 4
SUCCESS: int = 0
```

frost.client.objects module

```
class frost.client.objects.Memory
```

Bases: object

Stores information that needs to be passed around and easily accessible.

```
classmethod add_room_members (room_id: int, *members: Tuple[Dict[str, Union[str, int]]]) → None
    Store members within a room stored in Memory.rooms.
```

Parameters `room_id (int)` – The ID of the room to store the members in

```
classmethod add_rooms (*rooms: Tuple[Dict[str, Union[str, int]]]) → None
    Store rooms as frost.client.objects.Room objects in Memory.rooms.
```

Parameters `*rooms` – The rooms to store

```
classmethod get_room_member_changes () → Dict[str, Dict[int, frost.client.objects.Room]]
    Get changes in whether a members left or joined rooms.
```

Returns Room member changes

Return type Dict[str, Dict[int, 'Room']]

```
member_changes = {'joined': {}, 'left': {}}
    New members which have just joined a room or members who have just left the room, grouped by room.
```

```
classmethod remove_room_member (room_id: int, user_id: int) → None
    Remove a room member who left stored in a room.
```

Parameters

- `room_id (int)` – The room's ID the user left from
- `user_id (int)` – The ID of the user who left

```
rooms = {}
    The rooms in a server.
```

```
classmethod set_invite_code (room_id: int, invite_code: str) → None
    Store the invite code of a specific room.
```

Parameters

- `room_id (int)` – The ID of the room that the invite code corresponds to
- `invite_code (str)` – The invite code to store

```
class frost.client.objects.Room (id_: int, name: str, members: Dict[int, User] = {})
```

Bases: object

Represents a room in a server, storing information about one.

Parameters

- `id (int)` – The room's ID

- **name** (*str*) – The room’s name
- **members** (*Dict[int, 'User'], optional*) – The members a part of the room, defaults to list()

class `frost.client.objects.User` (*id_: int, username: str*)

Bases: `object`

Represents a user in a server, storing information about one.

Parameters

- **id** (*int*) – The user’s ID
- **username** (*str*) – The user’s username

Module contents

class `frost.client.FrostClient` (*ip: str = '127.0.0.1', port: int = 5555*)

Bases: `frost.client.socketio.base_client.BaseClient`

The Frost Client.

Parameters

- **ip** (*str, optional*) – The IP address of the server to connect to, defaults to ‘127.0.0.1’
- **port** (*int, optional*) – The port of the server to connect to, defaults to 5555

connect () → None

Connect to the server and begin listening for events.

create_room (*room_name: str, token: str, id_: str*) → None

Create a new room in a server.

Parameters

- **room_name** (*str*) – The name of the room to create
- **token** (*str*) – The user’s token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user’s ID, auto filled by `frost.client.auth.get_auth()`

get_invite_code (*room_id: int, token: str, id_: str*) → None

Get the invite code of a room in a server.

Parameters

- **room_id** (*int*) – The ID of the room to get an invite code from
- **token** (*str*) – The user’s token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user’s ID, auto filled by `frost.client.auth.get_auth()`

get_joined_rooms (*token: str, id_: str*) → None

Get all the joined rooms of the currently logged in user.

Parameters

- **token** (*str*) – The user’s token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user’s ID, auto filled by `frost.client.auth.get_auth()`

get_room_members (*room_id: int, token: str, id_: str*) → None

Get all the members of a specific room.

Parameters

- **room_id** (*int*) – The ID of the room to get the members of
- **token** (*str*) – The user’s token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user’s ID, auto filled by `frost.client.auth.get_auth()`

get_room_msgs (*room_id: int, token: str, id_: str*) → None

Get all messages from a specific room in a server.

Parameters

- **room_id** (*int*) – The ID of the room to get the messages from
- **token** (*str*) – The user’s token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user’s ID, auto filled by `frost.client.auth.get_auth()`

join_room (*invite_code: str, token: str, id_: str*) → None

Join a room in a server with an invite code.

Parameters

- **invite_code** (*str*) – The invite code the room to join
- **token** (*str*) – The user’s token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user’s ID, auto filled by `frost.client.auth.get_auth()`

leave_room (*room_id: int, token: str, id_: str*) → None

Leave a joined room in a server.

Parameters

- **room_id** (*int*) – The ID of the room to leave
- **token** (*str*) – The user’s token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user’s ID, auto filled by `frost.client.auth.get_auth()`

login (*username: str, password: str*) → None

Login to the server.

Parameters

- **username** (*str*) – The username of the account
- **password** (*str*) – The password of the account

register (*username: str, password: str*) → None

Register an account on the server.

Parameters

- **username** (*str*) – The desired username of the account
- **password** (*str*) – The desired password of the account

send_msg (*room_id: int, msg: str, token: str, id_: str*) → None

Send a message to other users on a server in a specific room.

Parameters

- **room_id** (*int*) – The ID of the room to send the message to
- **msg** (*str*) – The desired message to send
- **token** (*str*) – The user’s token, auto filled by `frost.client.auth.get_auth()`

- **id**(*str*) – The user’s ID, auto filled by `frost.client.auth.get_auth()`

class `frost.client.Status`

Bases: `enum.Enum`

The status enums

DUPLICATE_ROOM_NAME: `int` = 6

DUPLICATE_USERNAME: `int` = 3

EMPTY_ROOM_NAME: `int` = 5

INVALID_AUTH: `int` = 1

INVALID_INVITE: `int` = 7

PERMISSION_DENIED: `int` = 2

ROOM_NOT_FOUND: `int` = 4

SUCCESS: `int` = 0

class `frost.client.threaded`(**args: Any, **kwargs: Any*)

Bases: `object`

A decorator to thread a function/method.

`frost.client.get_auth`(*func: Callable*) → `Callable`

A decorator to get the saved auth token and id.

Parameters **func** (*Callable*) – The function that is being wrapped

Returns The inner execute function

Return type `Callable`

class `frost.client.Memory`

Bases: `object`

Stores information that needs to be passed around and easily accessible.

classmethod `add_room_members`(*room_id: int, *members: Tuple[Dict[str, Union[str, int]]]*) → `None`

Store members within a room stored in `Memory.rooms`.

Parameters **room_id** (*int*) – The ID of the room to store the members in

classmethod `add_rooms`(**rooms: Tuple[Dict[str, Union[str, int]]]*) → `None`

Store rooms as `frost.client.objects.Room` objects in `Memory.rooms`.

Parameters ***rooms** – The rooms to store

classmethod `get_room_member_changes`() → `Dict[str, Dict[int, frost.client.objects.Room]]`

Get changes in whether a members left or joined rooms.

Returns Room member changes

Return type `Dict[str, Dict[int, 'Room']]`

member_changes = {'joined': {}, 'left': {}}

New members which have just joined a room or members who have just left the room, grouped by room.

classmethod `remove_room_member`(*room_id: int, user_id: int*) → `None`

Remove a room member who left stored in a room.

Parameters

- **room_id** (*int*) – The room’s ID the user left from

- **user_id** (*int*) – The ID of the user who left

rooms = {}

The rooms in a server.

classmethod **set_invite_code** (*room_id: int, invite_code: str*) → None

Store the invite code of a specific room.

Parameters

- **room_id** (*int*) – The ID of the room that the invite code corresponds to
- **invite_code** (*str*) – The invite code to store

3.1.1.2 frost.ext package

Submodules

frost.ext.cog module

class `frost.ext.cog.Cog`

Bases: `object`

Children of this class and its methods are automatically routed and then handled by `frost.ext.handler.Handler`. Private methods of the children are ignored and not routed. All children methods are automatically run as static methods.

Raises `DirectCogInstanceError` – If this class is directly instantiated. This class can only be subclassed.

frost.ext.exceptions module

exception `frost.ext.exceptions.DirectCogInstanceError`

Bases: `Exception`

Raised if the Cog class is directly instantiated.

frost.ext.handler module

class `frost.ext.handler.Handler`

Bases: `object`

Handles incoming requests and executes methods according to the route path mapping.

handle (*data: Dict[str, Any], **kwargs*) → None

Handles the route and executes the resulting method.

Parameters **data** (*Dict[str, Any]*) – Data received from the server

Module contents

class `frost.ext.Cog`

Bases: `object`

Children of this class and its methods are automatically routed and then handled by `frost.ext.handler.Handler`. Private methods of the children are ignored and not routed. All children methods are automatically run as static methods.

Raises `DirectCogInstanceError` – If this class is directly instantiated. This class can only be subclassed.

class `frost.ext.Handler`

Bases: `object`

Handles incoming requests and executes methods according to the route path mapping.

handle (*data: Dict[str, Any], **kwargs*) → `None`

Handles the route and executes the resulting method.

Parameters *data* (`Dict[str, Any]`) – Data received from the server

3.1.1.3 frost.server package

Subpackages

frost.server.database package

Submodules

frost.server.database.db module

class `frost.server.database.db.Base` (***kwargs*)

Bases: `object`

The most base type

metadata = `MetaData(bind=None)`

`frost.server.database.db.engine` = `Engine(sqlite:///pyfrost.sqlite3)`

The SQLAlchemy engine.

`frost.server.database.db.init_db()` → `None`

Initializes a database and creates the pre-defined models.

`frost.server.database.db.managed_session()` → `Session`

A context manager for thread-safe database access. Automatically commits if no errors occur, else it is rolled back. Session is removed after use.

Yield An SQLAlchemy `scoped_session`

Return type `sqlalchemy.orm.Session`

frost.server.database.models module

```
class frost.server.database.models.Message (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    The Message model. __tablename__ = 'messages'

    id
        The message's ID.

    message
        The message's contents.

    room
        The room the message was sent in.

    room_id
        The ID of the room the message was sent in.

    timestamp
        The date and time the message was sent.

    user
        The user who sent the message.

    user_id
        The ID of the user who sent the message.

class frost.server.database.models.Room (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    The Room model. __tablename__ = 'rooms'

    id
        The room's ID.

    invite_code
        The room's invite code.

    messages
        The messages that have been sent in the room.

    name
        The room's name.

    owner_id
        The ID of the user who created the room.

    users
        The users who are have joined the room.

class frost.server.database.models.User (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    The User model. __tablename__ = 'users'

    id
        The user's ID.

    joined_rooms
        The different rooms the user has joined.

    messages
        The messages the user has sent.
```

password
The user's password.

token
The user's authentication token, used after login.

username
The user's username.

`frost.server.database.models.user_room_association = Table('user_room_association', MetaData(...))`
An association from the many-to-many relationship between users and rooms.

Module contents

`frost.server.database.init_db()` → None
Initializes a database and creates the pre-defined models.

`frost.server.database.managed_session()` → Session
A context manager for thread-safe database access. Automatically commits if no errors occur, else it is rolled back. Session is removed after use.

Yield An SQLAlchemy `scoped_session`

Return type `sqlalchemy.orm.Session`

class `frost.server.database.Base` (**kwargs)
Bases: object

The most base type

metadata = `MetaData(bind=None)`

class `frost.server.database.User` (**kwargs)
Bases: `sqlalchemy.ext.declarative.api.Base`

The User model. `__tablename__` = 'users'

id
The user's ID.

joined_rooms
The different rooms the user has joined.

messages
The messages the user has sent.

password
The user's password.

token
The user's authentication token, used after login.

username
The user's username.

class `frost.server.database.Room` (**kwargs)
Bases: `sqlalchemy.ext.declarative.api.Base`

The Room model. `__tablename__` = 'rooms'

id
The room's ID.

invite_code

The room's invite code.

messages

The messages that have been sent in the room.

name

The room's name.

owner_id

The ID of the user who created the room.

users

The users who are have joined the room.

```
class frost.server.database.Message (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base
    The Message model. __tablename__ = 'messages'

    id
        The message's ID.

    message
        The message's contents.

    room
        The room the message was sent in.

    room_id
        The ID of the room the message was sent in.

    timestamp
        The date and time the message was sent.

    user
        The user who sent the message.

    user_id
        The ID of the user who sent the message.
```

frost.server.socketio package

Submodules

frost.server.socketio.base_server module

```
class frost.server.socketio.base_server.BaseServer (ip: str = '127.0.0.1', port: int = 5555)
    Bases: object
```

A basic socket server to send and receive data from multiple clients. Assign self.func to a method with conn, addr as parameters to handle new user connections, as shown in [frost.server.server.FrostServer.on_user_connect\(\)](#)

Parameters

- **ip** (str, optional) – The IP address for the server to bind to, defaults to '127.0.0.1'
- **port** (int, optional) – The port for the server to bind to, defaults to 5555

recieve (*conn: socket.socket*) → Any

Receive data from a specific client.

Parameters **conn** (*socket.socket*) – The connected client socket

Returns The data received from the client

Return type Any

send (*conn: socket.socket, data: Any*) → None

Send data to a specific connected client.

Parameters

- **conn** (*socket.socket*) – The connected client socket
- **data** (*Any*) – The data to send to the client

start () → None

Starts the threaded, multi-client server.

```
class frost.server.socketio.base_server.ConnectionData (conn: Optional[socket.socket]
                                                         = None, addr: Optional[Tuple[str, int]] = None)
```

Bases: object

An object used to store connection data accross threads.

Parameters

- **conn** (*Optional[socket.socket]*) – The client’s connection, defaults to None
- **addr** (*Optional[Tuple[str, int]]*) – The client’s IP address and port, defaults to None

addr: **Optional[Tuple[str, int]] = None**

conn: **Optional[socket.socket] = None**

frost.server.socketio.utils module

```
class frost.server.socketio.utils.threaded (*args: Any, **kwargs: Any)
```

Bases: object

A decorator to thread a function/method.

Module contents

```
class frost.server.socketio.BaseServer (ip: str = '127.0.0.1', port: int = 5555)
```

Bases: object

A basic socket server to send and receive data from multiple clients. Assign self.func to a method with conn, addr as parameters to handle new user connections, as shown in `frost.server.server.FrostServer.on_user_connect()`

Parameters

- **ip** (*str, optional*) – The IP address for the server to bind to, defaults to ‘127.0.0.1’
- **port** (*int, optional*) – The port for the server to bind to, defaults to 5555

recieve (*conn: socket.socket*) → Any

Receive data from a specific client.

Parameters **conn** (*socket.socket*) – The connected client socket

Returns The data received from the client

Return type Any

send (*conn: socket.socket, data: Any*) → None

Send data to a specific connected client.

Parameters

- **conn** (*socket.socket*) – The connected client socket
- **data** (*Any*) – The data to send to the client

start () → None

Starts the threaded, multi-client server.

class `frost.server.socketio.threaded` (**args: Any, **kwargs: Any*)

Bases: object

A decorator to thread a function/method.

frost.server.storage package

Submodules

frost.server.storage.base module

class `frost.server.storage.base.Base`

Bases: object

The base model for data storage.

classmethod **add** (*item: Any*) → str

Adds an item under the given specific table.

Parameters **item** (*Any*) – The item to add

Raises **KeyError** – If an item under a given ID already exists

Returns The ID of the given item stored under the given table.

Return type str

static **commit** (*data: Dict[str, Any]*) → None

Commits and saves the data.

Parameters **data** (*Dict[str, Any]*) – The data to be saved

static **data** () → Dict[str, Any]

Gets the contents of the saved data.

Returns The contents of the saved data

Return type Dict[str, Any]

classmethod **entries** () → Dict[str, Any]

Get the entries under the specified table.

Returns The entries under the specified table

Return type Dict[str, Any]

classmethod search (*item: Any*) → Any
Searches the saved data for a specific item.

Parameters *item* (Any) – The item to search for

Returns The data found under the specific item, returns None if not found

Return type Any

classmethod update (*item: Any*) → None
Updates the given item's data. Does not care if the item already exists.

Parameters *item* (Any) – The item to update

class frost.server.storage.base.**Unique** (*data: Any*)
Bases: object

Ensures that the data stored is unique within its table.

Parameters *data* (Any) – The unique data to be stored in a table

frost.server.storage.defaults module

frost.server.storage.exceptions module

exception frost.server.storage.exceptions.**DuplicateValueError**
Bases: Exception

There was a duplicate value trying to be saved in a new entry within a specific table.

frost.server.storage.models module

class frost.server.storage.models.**Message** (*message: str, timestamp: str, from_user: Dict[str, str], id_: str = None*)

Bases: *frost.server.storage.base.Base*

The message model (__tablename__ = 'messages')

Parameters

- **message** (*str*) – The message sent from a user
- **timestamp** (*str*) – The timestamp the message was sent
- **from_user** (*Dict[str, str]*) – Information about who sent the message
- **id** (*str, optional*) – The message's ID, autofilled by *frost.server.storage.base._get_id()* if None

class frost.server.storage.models.**Room** (*name: str, owner_id: str, id_: str = None, members: List[Dict[str, str]] = []*)

Bases: *frost.server.storage.base.Base*

The room model (__tablename__ = 'rooms')

Parameters

- **name** (*str*) – The room's name
- **owner_id** (*str*) – The ID of the owner of the room

- **id** (*str*, *optional*) – The room’s ID, autofilled by `frost.server.storage.base._get_id()` if None
- **members** (*list*, *optional*) – The members of the room, defaults to []

class `frost.server.storage.models.User` (*username: str, password: str, id_: str = None, token: str = None*)

Bases: `frost.server.storage.base.Base`

The user model (`__tablename__ = 'users'`)

Parameters

- **username** (*str*) – The user’s username
- **password** (*str*) – The user’s password
- **id** (*str*, *optional*) – The user’s ID, autofilled by `frost.server.storage.base._get_id()` if None
- **token** (*str*, *optional*) – The user’s token, defaults to None

Module contents

class `frost.server.storage.Base`

Bases: `object`

The base model for data storage.

classmethod `add` (*item: Any*) → `str`

Adds an item under the given specific table.

Parameters **item** (*Any*) – The item to add

Raises **KeyError** – If an item under a given ID already exists

Returns The ID of the given item stored under the given table.

Return type `str`

static `commit` (*data: Dict[str, Any]*) → `None`

Commits and saves the data.

Parameters **data** (*Dict[str, Any]*) – The data to be saved

static `data` () → `Dict[str, Any]`

Gets the contents of the saved data.

Returns The contents of the saved data

Return type `Dict[str, Any]`

classmethod `entries` () → `Dict[str, Any]`

Get the entries under the specified table.

Returns The entries under the specified table

Return type `Dict[str, Any]`

classmethod `search` (*item: Any*) → `Any`

Searches the saved data for a specific item.

Parameters **item** (*Any*) – The item to search for

Returns The data found under the specific item, returns None if not found

Return type Any

classmethod `update(item: Any) → None`

Updates the given item's data. Does not care if the item already exists.

Parameters `item (Any)` – The item to update

class `frost.server.storage.User(username: str, password: str, id_: str = None, token: str = None)`

Bases: `frost.server.storage.base.Base`

The user model (`__tablename__ = 'users'`)

Parameters

- **username** (`str`) – The user's username
- **password** (`str`) – The user's password
- **id** (`str, optional`) – The user's ID, autofilled by `frost.server.storage.base._get_id()` if None
- **token** (`str, optional`) – The user's token, defaults to None

class `frost.server.storage.Room(name: str, owner_id: str, id_: str = None, members: List[Dict[str, str]] = [])`

Bases: `frost.server.storage.base.Base`

The room model (`__tablename__ = 'rooms'`)

Parameters

- **name** (`str`) – The room's name
- **owner_id** (`str`) – The ID of the owner of the room
- **id** (`str, optional`) – The room's ID, autofilled by `frost.server.storage.base._get_id()` if None
- **members** (`list, optional`) – The members of the room, defaults to []

class `frost.server.storage.Message(message: str, timestamp: str, from_user: Dict[str, str], id_: str = None)`

Bases: `frost.server.storage.base.Base`

The message model (`__tablename__ = 'messages'`)

Parameters

- **message** (`str`) – The message sent from a user
- **timestamp** (`str`) – The timestamp the message was sent
- **from_user** (`Dict[str, str]`) – Information about who sent the message
- **id** (`str, optional`) – The message's ID, autofilled by `frost.server.storage.base._get_id()` if None

exception `frost.server.storage.DuplicateValueError`

Bases: `Exception`

There was a duplicate value trying to be saved in a new entry within a specific table.

Submodules

frost.server.auth module

`frost.server.auth.auth_required` (*func: Callable*) → *Callable*

A decorator to ensure a client is logged in with valid authentication before running the wrapped function. Automatically passes through the user's ID (`id_`) and token (`token`) as arguments.

Parameters `func` (*Callable*) – The function being wrapped

Returns The inner execute function

Return type *Callable*

frost.server.cogs module

class `frost.server.cogs.Auth`

Bases: `frost.ext.cog.Cog`

Deals with user authentication. `route='authentication'`

login (***kwargs: Any*) → *None*

Logs in a user with the given username and password.

Parameters `data` (*Dict[str, Any]*) – Data received from the client

register (***kwargs: Any*) → *None*

Registers a new user with the given username and password.

Parameters `data` (*Dict[str, Any]*) – Data received from the client

`route = 'authentication'`

class `frost.server.cogs.Msgs`

Bases: `frost.ext.cog.Cog`

Deals with user messages. `route='messages'`

get_room_msgs (*token: str, id_: str, max_: int = 100, **kwargs: Any*) → *None*

Gets up to `max_` previous messages from a specific room.

Parameters

- `data` (*Dict[str, Any]*) – Data received from the client
- `max` (*int, optional*) – The maximum number of messages to get from a room, defaults to 50
- `token` (*str*) – The user's token, autofilled by `frost.server.auth.auth_required()`
- `id` (*str*) – The user's ID, autofilled by `frost.server.auth.auth_required()`

`route = 'messages'`

send_msg (*token: str, id_: str, **kwargs: Any*) → *None*

Saves the message received from a client and sends it off to other users in the room.

Parameters

- `data` (*Dict[str, Any]*) – Data received from a client

- **token** (*str*) – The user's token, autofilled by `frost.server.auth.auth_required()`
- **id** (*str*) – The user's ID, autofilled by `frost.server.auth.auth_required()`

class `frost.server.cogs.Rooms`

Bases: `frost.ext.cog.Cog`

Deals with room within a server. `route='rooms'`

create (*token: str, id_: str, **kwargs: Any*) → None
Creates a new room in the server.

Parameters

- **data** (*Dict[str, Any]*) – Data received from a client
- **token** (*str*) – The user's token, autofilled by `frost.server.auth.auth_required()`
- **id** (*str*) – The user's ID, autofilled by `frost.server.auth.auth_required()`

get_all_joined (*token: str, id_: str, **kwargs: Any*) → None
Get the all the rooms a specific user has joined.

Parameters

- **data** (*Dict[str, Any]*) – Data received from a client
- **token** (*str*) – The user's token, autofilled by `frost.server.auth.auth_required()`
- **id** (*str*) – The user's ID, autofilled by `frost.server.auth.auth_required()`

get_invite_code (*token: str, id_: str, **kwargs: Any*) → None
Get the invite code of a specific room in the server. Only an owner can request for their own room's invite code.

Parameters

- **data** (*Dict[str, Any]*) – Data received from a client
- **token** (*str*) – The user's token, autofilled by `frost.server.auth.auth_required()`
- **id** (*str*) – The user's ID, autofilled by `frost.server.auth.auth_required()`

get_members (*token: str, id_: str, **kwargs: Any*) → None
Get all the members of a specific room the user has joined.

Parameters

- **data** (*Dict[str, Any]*) – Data received from a client
- **token** (*str*) – The user's token, autofilled by `frost.server.auth.auth_required()`
- **id** (*str*) – The user's ID, autofilled by `frost.server.auth.auth_required()`

join (*token: str, id_: str, **kwargs: Any*) → None
Join an existing room within the server with an invite code.

Parameters

- **data** (*Dict[str, Any]*) – Data received from a client

- **token** (*str*) – The user’s token, autofilled by `frost.server.auth.auth_required()`
- **id** (*str*) – The user’s ID, autofilled by `frost.server.auth.auth_required()`

leave (*token: str, id_: str, **kwargs: Any*) → None
Leave a specific room within the server.

Parameters

- **data** (*Dict[str, Any]*) – Data received from a client
- **token** (*str*) – The user’s token, autofilled by `frost.server.auth.auth_required()`
- **id** (*str*) – The user’s ID, autofilled by `frost.server.auth.auth_required()`

route = 'rooms'

frost.server.headers module

```
class frost.server.headers.Status
    Bases: enum.Enum
    The status enums
    DUPLICATE_ROOM_NAME: int = 6
    DUPLICATE_USERNAME: int = 3
    EMPTY_ROOM_NAME: int = 5
    INVALID_AUTH: int = 1
    INVALID_INVITE: int = 7
    PERMISSION_DENIED: int = 2
    ROOM_NOT_FOUND: int = 4
    SUCCESS: int = 0
```

frost.server.logger module

The logger for the server (logger).

frost.server.objects module

```
class frost.server.objects.Memory
    Bases: object
    Stores information that needs to be passed around and easily accessible.
    all_users = {}
        All connected users.
    logged_in_users = {}
        All logged in users.
```

```
class frost.server.objects.UserObj (addr: Tuple[str, int], conn: socket.socket, id_: Optional[int] = None, username: Optional[str] = None)
```

Bases: object

Represents a user.

Parameters

- **addr** (*Tuple[str, int]*) – The IP address and port of the connected user
- **conn** (*'socket.socket'*) – The socket instance of the connected user
- **id** (*Optional[int]*) – The user's ID, defaults to None
- **username** (*Optional[str]*) – The user's username, defaults to None

property is_logged_in

Returns whether or not the user is logged in.

Returns Whether or not the user is logged in

Return type bool

```
login (id_: int, username: str) → None
```

Sets the user's ID and username, "logging them in".

Parameters

- **id** (*int*) – The user's ID
- **username** (*str*) – The user's username

frost.server.server module

```
class frost.server.server.FrostServer (file: str)
```

Bases: *frost.server.socketio.base_server.BaseServer*

The Frost server.

Parameters **file** (*str*) – The `__file__` of the file this is imported in

```
on_user_connect (conn: socket.socket, addr: Tuple[str, int]) → None
```

Handles the connection of a client and executes tasks accordingly.

Parameters

- **conn** (*'socket.socket'*) – The client's connection
- **addr** (*Tuple[str, int]*) – The user's IP address and port

```
run (ip: str = '127.0.0.1', port: int = 5555) → None
```

Runs the FrostServer.

Parameters

- **ip** (*str, optional*) – The IP for the server to bind to, defaults to '127.0.0.1'
- **port** (*int, optional*) – The port for the server to bind to, defaults to 5555

```
frost.server.server.send_partial (send_func: Callable, conn: socket.socket) → Callable
```

A partial function to auto fill the `conn` parameter of *frost.server.socketio.base_server.BaseServer()*

Parameters

- **send_func** (*Callable*) – The send function that sends data to the client

- **conn** (*socket.socket*) – A specific client’s connection

Returns The inner execute function

Return type Callable

Module contents

class `frost.server.FrostServer` (*file: str*)

Bases: `frost.server.socketio.base_server.BaseServer`

The Frost server.

Parameters **file** (*str*) – The `__file__` of the file this is imported in

on_user_connect (*conn: socket.socket, addr: Tuple[str, int]*) → None

Handles the connection of a client and executes tasks accordingly.

Parameters

- **conn** (*'socket.socket'*) – The client’s connection
- **addr** (*Tuple[str, int]*) – The user’s IP address and port

run (*ip: str = '127.0.0.1', port: int = 5555*) → None

Runs the FrostServer.

Parameters

- **ip** (*str, optional*) – The IP for the server to bind to, defaults to ‘127.0.0.1’
- **port** (*int, optional*) – The port for the server to bind to, defaults to 5555

class `frost.server.Status`

Bases: `enum.Enum`

The status enums

DUPLICATE_ROOM_NAME: `int = 6`

DUPLICATE_USERNAME: `int = 3`

EMPTY_ROOM_NAME: `int = 5`

INVALID_AUTH: `int = 1`

INVALID_INVITE: `int = 7`

PERMISSION_DENIED: `int = 2`

ROOM_NOT_FOUND: `int = 4`

SUCCESS: `int = 0`

class `frost.server.threaded` (**args: Any, **kwargs: Any*)

Bases: `object`

A decorator to thread a function/method.

`frost.server.auth_required` (*func: Callable*) → Callable

A decorator to ensure a client is logged in with valid authentication before running the wrapped function. Automatically passes through the user’s ID (`id_`) and token (`token`) as arguments.

Parameters **func** (*Callable*) – The function being wrapped

Returns The inner execute function

Return type Callable

class `frost.server.Memory`

Bases: `object`

Stores information that needs to be passed around and easily accessible.

all_users = {}

All connected users.

logged_in_users = {}

All logged in users.

3.1.2 Module contents

class `frost.FrostServer` (*file: str*)

Bases: `frost.server.socketio.base_server.BaseServer`

The Frost server.

Parameters **file** (*str*) – The `__file__` of the file this is imported in

on_user_connect (*conn: socket.socket, addr: Tuple[str, int]*) → None

Handles the connection of a client and executes tasks accordingly.

Parameters

- **conn** (*'socket.socket'*) – The client's connection
- **addr** (*Tuple[str, int]*) – The user's IP address and port

run (*ip: str = '127.0.0.1', port: int = 5555*) → None

Runs the FrostServer.

Parameters

- **ip** (*str, optional*) – The IP for the server to bind to, defaults to '127.0.0.1'
- **port** (*int, optional*) – The port for the server to bind to, defaults to 5555

class `frost.FrostClient` (*ip: str = '127.0.0.1', port: int = 5555*)

Bases: `frost.client.socketio.base_client.BaseClient`

The Frost Client.

Parameters

- **ip** (*str, optional*) – The IP address of the server to connect to, defaults to '127.0.0.1'
- **port** (*int, optional*) – The port of the server to connect to, defaults to 5555

connect () → None

Connect to the server and begin listening for events.

create_room (*room_name: str, token: str, id_: str*) → None

Create a new room in a server.

Parameters

- **room_name** (*str*) – The name of the room to create
- **token** (*str*) – The user's token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user's ID, auto filled by `frost.client.auth.get_auth()`

get_invite_code (*room_id: int, token: str, id_: str*) → None

Get the invite code of a room in a server.

Parameters

- **room_id** (*int*) – The ID of the room to get an invite code from
- **token** (*str*) – The user’s token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user’s ID, auto filled by `frost.client.auth.get_auth()`

get_joined_rooms (*token: str, id_: str*) → None

Get all the joined rooms of the currently logged in user.

Parameters

- **token** (*str*) – The user’s token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user’s ID, auto filled by `frost.client.auth.get_auth()`

get_room_members (*room_id: int, token: str, id_: str*) → None

Get all the members of a specific room.

Parameters

- **room_id** (*int*) – The ID of the room to get the members of
- **token** (*str*) – The user’s token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user’s ID, auto filled by `frost.client.auth.get_auth()`

get_room_msgs (*room_id: int, token: str, id_: str*) → None

Get all messages from a specific room in a server.

Parameters

- **room_id** (*int*) – The ID of the room to get the messages from
- **token** (*str*) – The user’s token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user’s ID, auto filled by `frost.client.auth.get_auth()`

join_room (*invite_code: str, token: str, id_: str*) → None

Join a room in a server with an invite code.

Parameters

- **invite_code** (*str*) – The invite code the room to join
- **token** (*str*) – The user’s token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user’s ID, auto filled by `frost.client.auth.get_auth()`

leave_room (*room_id: int, token: str, id_: str*) → None

Leave a joined room in a server.

Parameters

- **room_id** (*int*) – The ID of the room to leave
- **token** (*str*) – The user’s token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user’s ID, auto filled by `frost.client.auth.get_auth()`

login (*username: str, password: str*) → None

Login to the server.

Parameters

- **username** (*str*) – The username of the account
- **password** (*str*) – The password of the account

register (*username: str, password: str*) → None

Register an account on the server.

Parameters

- **username** (*str*) – The desired username of the account
- **password** (*str*) – The desired password of the account

send_msg (*room_id: int, msg: str, token: str, id_: str*) → None

Send a message to other users on a server in a specific room.

Parameters

- **room_id** (*int*) – The ID of the room to send the message to
- **msg** (*str*) – The desired message to send
- **token** (*str*) – The user's token, auto filled by `frost.client.auth.get_auth()`
- **id** (*str*) – The user's ID, auto filled by `frost.client.auth.get_auth()`

class `frost.Cog`

Bases: `object`

Children of this class and its methods are automatically routed and then handled by `frost.ext.handler.Handler`. Private methods of the children are ignored and not routed. All children methods are automatically run as static methods.

Raises `DirectCogInstanceError` – If this class is directly instantiated. This class can only be subclassed.

class `frost.Handler`

Bases: `object`

Handles incoming requests and executes methods according to the route path mapping.

handle (*data: Dict[str, Any], **kwargs*) → None

Handles the route and executes the resulting method.

Parameters **data** (*Dict[str, Any]*) – Data received from the server

PYTHON MODULE INDEX

f

- frost, 35
- frost.client, 17
 - auth, 13
 - client, 14
 - events, 10
 - cogs, 7
 - events, 9
 - headers, 15
 - objects, 16
 - socketio, 13
 - base_client, 12
 - utils, 12
- frost.ext, 21
 - cog, 20
 - exceptions, 20
 - handler, 20
- frost.server, 34
 - auth, 30
 - cogs, 30
 - database, 23
 - db, 21
 - models, 22
 - headers, 32
 - logger, 32
 - objects, 32
 - server, 33
 - socketio, 25
 - base_server, 24
 - utils, 25
 - storage, 28
 - base, 26
 - defaults, 27
 - exceptions, 27
 - models, 27

A

add() (*frost.server.storage.Base class method*), 28
 add() (*frost.server.storage.base.Base class method*), 26
 add_new_msgs() (*frost.client.events.events.Messages class method*), 9
 add_new_msgs() (*frost.client.events.Messages class method*), 11
 add_room_members() (*frost.client.Memory class method*), 19
 add_room_members() (*frost.client.objects.Memory class method*), 16
 add_rooms() (*frost.client.Memory class method*), 19
 add_rooms() (*frost.client.objects.Memory class method*), 16
 addr (*frost.server.socketio.base_server.ConnectionData attribute*), 25
 all (*frost.client.events.events.Messages attribute*), 9
 all (*frost.client.events.Messages attribute*), 11
 all_users (*frost.server.Memory attribute*), 35
 all_users (*frost.server.objects.Memory attribute*), 32
 Auth (*class in frost.client.events*), 10
 Auth (*class in frost.client.events.cogs*), 7
 Auth (*class in frost.server.cogs*), 30
 auth_required() (*in module frost.server*), 34
 auth_required() (*in module frost.server.auth*), 30

B

Base (*class in frost.server.database*), 23
 Base (*class in frost.server.database.db*), 21
 Base (*class in frost.server.storage*), 28
 Base (*class in frost.server.storage.base*), 26
 BaseClient (*class in frost.client.socketio*), 13
 BaseClient (*class in frost.client.socketio.base_client*), 12
 BaseServer (*class in frost.server.socketio*), 25
 BaseServer (*class in frost.server.socketio.base_server*), 24

C

clear() (*frost.client.events.events.Messages class method*), 9
 clear() (*frost.client.events.Messages class method*), 11

close() (*frost.client.socketio.base_client.BaseClient method*), 12
 close() (*frost.client.socketio.BaseClient method*), 13
 Cog (*class in frost*), 37
 Cog (*class in frost.ext*), 21
 Cog (*class in frost.ext.cog*), 20
 commit() (*frost.server.storage.Base static method*), 28
 commit() (*frost.server.storage.base.Base static method*), 26
 conn (*frost.server.socketio.base_server.ConnectionData attribute*), 25
 connect() (*frost.client.client.FrostClient method*), 14
 connect() (*frost.client.FrostClient method*), 17
 connect() (*frost.client.socketio.base_client.BaseClient method*), 12
 connect() (*frost.client.socketio.BaseClient method*), 13
 connect() (*frost.FrostClient method*), 35
 ConnectionData (*class in frost.server.socketio.base_server*), 25
 create() (*frost.server.cogs.Rooms method*), 31
 create_room (*frost.client.events.events.EventStatus attribute*), 9
 create_room (*frost.client.events.EventStatus attribute*), 11
 create_room() (*frost.client.client.FrostClient method*), 14
 create_room() (*frost.client.FrostClient method*), 17
 create_room() (*frost.FrostClient method*), 35

D

data() (*frost.server.storage.Base static method*), 28
 data() (*frost.server.storage.base.Base static method*), 26
 DirectCogInstanceError, 20
 DUPLICATE_ROOM_NAME (*frost.client.headers.Status attribute*), 15
 DUPLICATE_ROOM_NAME (*frost.client.Status attribute*), 19
 DUPLICATE_ROOM_NAME (*frost.server.headers.Status attribute*), 32

`DUPLICATE_ROOM_NAME` (*frost.server.Status attribute*), 34
`DUPLICATE_USERNAME` (*frost.client.headers.Status attribute*), 15
`DUPLICATE_USERNAME` (*frost.client.Status attribute*), 19
`DUPLICATE_USERNAME` (*frost.server.headers.Status attribute*), 32
`DUPLICATE_USERNAME` (*frost.server.Status attribute*), 34
`DuplicateValueError`, 27, 29

E

`EMPTY_ROOM_NAME` (*frost.client.headers.Status attribute*), 15
`EMPTY_ROOM_NAME` (*frost.client.Status attribute*), 19
`EMPTY_ROOM_NAME` (*frost.server.headers.Status attribute*), 32
`EMPTY_ROOM_NAME` (*frost.server.Status attribute*), 34
`engine` (*in module frost.server.database.db*), 21
`entries()` (*frost.server.storage.Base class method*), 28
`entries()` (*frost.server.storage.base.Base class method*), 26
`EventStatus` (*class in frost.client.events*), 11
`EventStatus` (*class in frost.client.events.events*), 9

F

`frost` (*module*), 35
`frost.client` (*module*), 17
`frost.client.auth` (*module*), 13
`frost.client.client` (*module*), 14
`frost.client.events` (*module*), 10
`frost.client.events.cogs` (*module*), 7
`frost.client.events.events` (*module*), 9
`frost.client.headers` (*module*), 15
`frost.client.objects` (*module*), 16
`frost.client.socketio` (*module*), 13
`frost.client.socketio.base_client` (*module*), 12
`frost.client.socketio.utils` (*module*), 12
`frost.ext` (*module*), 21
`frost.ext.cog` (*module*), 20
`frost.ext.exceptions` (*module*), 20
`frost.ext.handler` (*module*), 20
`frost.server` (*module*), 34
`frost.server.auth` (*module*), 30
`frost.server.cogs` (*module*), 30
`frost.server.database` (*module*), 23
`frost.server.database.db` (*module*), 21
`frost.server.database.models` (*module*), 22
`frost.server.headers` (*module*), 32
`frost.server.logger` (*module*), 32
`frost.server.objects` (*module*), 32
`frost.server.server` (*module*), 33

`frost.server.socketio` (*module*), 25
`frost.server.socketio.base_server` (*module*), 24
`frost.server.socketio.utils` (*module*), 25
`frost.server.storage` (*module*), 28
`frost.server.storage.base` (*module*), 26
`frost.server.storage.defaults` (*module*), 27
`frost.server.storage.exceptions` (*module*), 27
`frost.server.storage.models` (*module*), 27
`FrostClient` (*class in frost*), 35
`FrostClient` (*class in frost.client*), 17
`FrostClient` (*class in frost.client.client*), 14
`FrostServer` (*class in frost*), 35
`FrostServer` (*class in frost.server*), 34
`FrostServer` (*class in frost.server.server*), 33

G

`get_all_joined()` (*frost.server.cogs.Rooms method*), 31
`get_auth()` (*in module frost.client*), 19
`get_auth()` (*in module frost.client.auth*), 13
`get_auth_token` (*in module frost.client.auth*), 13
`get_id` (*in module frost.client.auth*), 13
`get_invite_code` (*frost.client.events.events.EventStatus attribute*), 9
`get_invite_code` (*frost.client.events.EventStatus attribute*), 11
`get_invite_code()` (*frost.client.client.FrostClient method*), 14
`get_invite_code()` (*frost.client.FrostClient method*), 17
`get_invite_code()` (*frost.FrostClient method*), 35
`get_invite_code()` (*frost.server.cogs.Rooms method*), 31
`get_joined_rooms` (*frost.client.events.events.EventStatus attribute*), 9
`get_joined_rooms` (*frost.client.events.EventStatus attribute*), 11
`get_joined_rooms()` (*frost.client.client.FrostClient method*), 14
`get_joined_rooms()` (*frost.client.FrostClient method*), 17
`get_joined_rooms()` (*frost.FrostClient method*), 36
`get_members()` (*frost.server.cogs.Rooms method*), 31
`get_new_msgs()` (*frost.client.events.events.Messages class method*), 9
`get_new_msgs()` (*frost.client.events.Messages class method*), 11
`get_room_member_changes()` (*frost.client.Memory class method*), 19
`get_room_member_changes()` (*frost.client.objects.Memory class method*), 16

get_room_members (*frost.client.events.events.EventStatus attribute*), 9
 get_room_members (*frost.client.events.EventStatus attribute*), 11
 get_room_members () (*frost.client.client.FrostClient method*), 14
 get_room_members () (*frost.client.FrostClient method*), 17
 get_room_members () (*frost.FrostClient method*), 36
 get_room_msgs (*frost.client.events.events.EventStatus attribute*), 9
 get_room_msgs (*frost.client.events.EventStatus attribute*), 11
 get_room_msgs () (*frost.client.client.FrostClient method*), 14
 get_room_msgs () (*frost.client.FrostClient method*), 18
 get_room_msgs () (*frost.FrostClient method*), 36
 get_room_msgs () (*frost.server.cogs.Msgs method*), 30
 get_status () (*frost.client.events.events.EventStatus class method*), 9
 get_status () (*frost.client.events.EventStatus class method*), 11

H

handle () (*frost.ext.Handler method*), 21
 handle () (*frost.ext.handler.Handler method*), 20
 handle () (*frost.Handler method*), 37
 Handler (*class in frost*), 37
 Handler (*class in frost.ext*), 21
 Handler (*class in frost.ext.handler*), 20

I

id (*frost.server.database.Message attribute*), 24
 id (*frost.server.database.models.Message attribute*), 22
 id (*frost.server.database.models.Room attribute*), 22
 id (*frost.server.database.models.User attribute*), 22
 id (*frost.server.database.Room attribute*), 23
 id (*frost.server.database.User attribute*), 23
 init_db () (*in module frost.server.database*), 23
 init_db () (*in module frost.server.database.db*), 21
 INVALID_AUTH (*frost.client.headers.Status attribute*), 15
 INVALID_AUTH (*frost.client.Status attribute*), 19
 INVALID_AUTH (*frost.server.headers.Status attribute*), 32
 INVALID_AUTH (*frost.server.Status attribute*), 34
 INVALID_INVITE (*frost.client.headers.Status attribute*), 15
 INVALID_INVITE (*frost.client.Status attribute*), 19
 INVALID_INVITE (*frost.server.headers.Status attribute*), 32
 INVALID_INVITE (*frost.server.Status attribute*), 34

invite_code (*frost.server.database.models.Room attribute*), 22
 invite_code (*frost.server.database.Room attribute*), 23
 is_logged_in () (*frost.server.objects.UserObj property*), 33

J

join () (*frost.server.cogs.Rooms method*), 31
 join_room (*frost.client.events.events.EventStatus attribute*), 9
 join_room (*frost.client.events.EventStatus attribute*), 11
 join_room () (*frost.client.client.FrostClient method*), 14
 join_room () (*frost.client.FrostClient method*), 18
 join_room () (*frost.FrostClient method*), 36
 joined_rooms (*frost.server.database.models.User attribute*), 22
 joined_rooms (*frost.server.database.User attribute*), 23

L

leave () (*frost.server.cogs.Rooms method*), 32
 leave_room (*frost.client.events.events.EventStatus attribute*), 9
 leave_room (*frost.client.events.EventStatus attribute*), 12
 leave_room () (*frost.client.client.FrostClient method*), 15
 leave_room () (*frost.client.FrostClient method*), 18
 leave_room () (*frost.FrostClient method*), 36
 logged_in_users (*frost.server.Memory attribute*), 35
 logged_in_users (*frost.server.objects.Memory attribute*), 32
 login (*frost.client.events.events.EventStatus attribute*), 9
 login (*frost.client.events.EventStatus attribute*), 12
 login () (*frost.client.client.FrostClient method*), 15
 login () (*frost.client.FrostClient method*), 18
 login () (*frost.FrostClient method*), 36
 login () (*frost.server.cogs.Auth method*), 30
 login () (*frost.server.objects.UserObj method*), 33

M

managed_session () (*in module frost.server.database*), 23
 managed_session () (*in module frost.server.database.db*), 21
 member_changes (*frost.client.Memory attribute*), 19
 member_changes (*frost.client.objects.Memory attribute*), 16
 Memory (*class in frost.client*), 19
 Memory (*class in frost.client.objects*), 16

Memory (class in *frost.server*), 35
Memory (class in *frost.server.objects*), 32
Message (class in *frost.server.database*), 24
Message (class in *frost.server.database.models*), 22
Message (class in *frost.server.storage*), 29
Message (class in *frost.server.storage.models*), 27
message (*frost.server.database.Message* attribute), 24
message (*frost.server.database.models.Message* attribute), 22
Messages (class in *frost.client.events*), 11
Messages (class in *frost.client.events.events*), 9
messages (*frost.server.database.models.Room* attribute), 22
messages (*frost.server.database.models.User* attribute), 22
messages (*frost.server.database.Room* attribute), 24
messages (*frost.server.database.User* attribute), 23
metadata (*frost.server.database.Base* attribute), 23
metadata (*frost.server.database.db.Base* attribute), 21
Msgs (class in *frost.client.events*), 10
Msgs (class in *frost.client.events.cogs*), 7
Msgs (class in *frost.server.cogs*), 30

N

name (*frost.server.database.models.Room* attribute), 22
name (*frost.server.database.Room* attribute), 24
new () (*frost.client.events.cogs.Msgs* method), 7
new () (*frost.client.events.Msgs* method), 10
new_room_member () (*frost.client.events.cogs.Rooms* method), 8
new_room_member () (*frost.client.events.Rooms* method), 10

O

on_user_connect () (*frost.FrostServer* method), 35
on_user_connect () (*frost.server.FrostServer* method), 34
on_user_connect () (*frost.server.server.FrostServer* method), 33
owner_id (*frost.server.database.models.Room* attribute), 22
owner_id (*frost.server.database.Room* attribute), 24

P

password (*frost.server.database.models.User* attribute), 22
password (*frost.server.database.User* attribute), 23
PERMISSION_DENIED (*frost.client.headers.Status* attribute), 16
PERMISSION_DENIED (*frost.client.Status* attribute), 19
PERMISSION_DENIED (*frost.server.headers.Status* attribute), 32

PERMISSION_DENIED (*frost.server.Status* attribute), 34
post_all_joined () (*frost.client.events.cogs.Rooms* method), 8
post_all_joined () (*frost.client.events.Rooms* method), 10
post_create () (*frost.client.events.cogs.Rooms* method), 8
post_create () (*frost.client.events.Rooms* method), 10
post_invite_code () (*frost.client.events.cogs.Rooms* method), 8
post_invite_code () (*frost.client.events.Rooms* method), 10
post_join () (*frost.client.events.cogs.Rooms* method), 8
post_join () (*frost.client.events.Rooms* method), 10
post_leave () (*frost.client.events.cogs.Rooms* method), 8
post_leave () (*frost.client.events.Rooms* method), 11
post_login () (*frost.client.events.Auth* method), 10
post_login () (*frost.client.events.cogs.Auth* method), 7
post_members () (*frost.client.events.cogs.Rooms* method), 8
post_members () (*frost.client.events.Rooms* method), 11
post_new () (*frost.client.events.cogs.Msgs* method), 7
post_new () (*frost.client.events.Msgs* method), 10
post_register () (*frost.client.events.Auth* method), 10
post_register () (*frost.client.events.cogs.Auth* method), 7
post_room () (*frost.client.events.cogs.Msgs* method), 8
post_room () (*frost.client.events.Msgs* method), 10

R

recieve () (*frost.client.socketio.base_client.BaseClient* method), 12
recieve () (*frost.client.socketio.BaseClient* method), 13
recieve () (*frost.server.socketio.base_server.BaseServer* method), 24
recieve () (*frost.server.socketio.BaseServer* method), 25
register (*frost.client.events.events.EventStatus* attribute), 9
register (*frost.client.events.EventStatus* attribute), 12
register () (*frost.client.client.FrostClient* method), 15
register () (*frost.client.FrostClient* method), 18
register () (*frost.FrostClient* method), 37
register () (*frost.server.cogs.Auth* method), 30

remove_room_member() (frost.client.events.cogs.Rooms method), 8
 remove_room_member() (frost.client.events.Rooms method), 11
 remove_room_member() (frost.client.Memory class method), 19
 remove_room_member() (frost.client.objects.Memory class method), 16
 Room (class in frost.client.objects), 16
 Room (class in frost.server.database), 23
 Room (class in frost.server.database.models), 22
 Room (class in frost.server.storage), 29
 Room (class in frost.server.storage.models), 27
 room (frost.server.database.Message attribute), 24
 room (frost.server.database.models.Message attribute), 22
 room_id (frost.server.database.Message attribute), 24
 room_id (frost.server.database.models.Message attribute), 22
 ROOM_NOT_FOUND (frost.client.headers.Status attribute), 16
 ROOM_NOT_FOUND (frost.client.Status attribute), 19
 ROOM_NOT_FOUND (frost.server.headers.Status attribute), 32
 ROOM_NOT_FOUND (frost.server.Status attribute), 34
 Rooms (class in frost.client.events), 10
 Rooms (class in frost.client.events.cogs), 8
 Rooms (class in frost.server.cogs), 31
 rooms (frost.client.Memory attribute), 20
 rooms (frost.client.objects.Memory attribute), 16
 route (frost.client.events.Auth attribute), 10
 route (frost.client.events.cogs.Auth attribute), 7
 route (frost.client.events.cogs.Msgs attribute), 8
 route (frost.client.events.cogs.Rooms attribute), 8
 route (frost.client.events.Msgs attribute), 10
 route (frost.client.events.Rooms attribute), 11
 route (frost.server.cogs.Auth attribute), 30
 route (frost.server.cogs.Msgs attribute), 30
 route (frost.server.cogs.Rooms attribute), 32
 run() (frost.FrostServer method), 35
 run() (frost.server.FrostServer method), 34
 run() (frost.server.server.FrostServer method), 33

S

search() (frost.server.storage.Base class method), 28
 search() (frost.server.storage.base.Base class method), 27
 send() (frost.client.socketio.base_client.BaseClient method), 12
 send() (frost.client.socketio.BaseClient method), 13
 send() (frost.server.socketio.base_server.BaseServer method), 25
 send() (frost.server.socketio.BaseServer method), 26
 send_msg (frost.client.events.events.EventStatus attribute), 9
 send_msg (frost.client.events.EventStatus attribute), 12
 send_msg() (frost.client.client.FrostClient method), 15
 send_msg() (frost.client.FrostClient method), 18
 send_msg() (frost.FrostClient method), 37
 send_msg() (frost.server.cogs.Msgs method), 30
 send_partial() (in module frost.server.server), 33
 set_invite_code() (frost.client.Memory class method), 20
 set_invite_code() (frost.client.objects.Memory class method), 16
 start() (frost.server.socketio.base_server.BaseServer method), 25
 start() (frost.server.socketio.BaseServer method), 26
 Status (class in frost.client), 19
 Status (class in frost.client.headers), 15
 Status (class in frost.server), 34
 Status (class in frost.server.headers), 32
 SUCCESS (frost.client.headers.Status attribute), 16
 SUCCESS (frost.client.Status attribute), 19
 SUCCESS (frost.server.headers.Status attribute), 32
 SUCCESS (frost.server.Status attribute), 34

T

threaded (class in frost.client), 19
 threaded (class in frost.client.socketio), 13
 threaded (class in frost.client.socketio.utils), 12
 threaded (class in frost.server), 34
 threaded (class in frost.server.socketio), 26
 threaded (class in frost.server.socketio.utils), 25
 timestamp (frost.server.database.Message attribute), 24
 timestamp (frost.server.database.models.Message attribute), 22
 token (frost.server.database.models.User attribute), 23
 token (frost.server.database.User attribute), 23

U

Unique (class in frost.server.storage.base), 27
 update() (frost.server.storage.Base class method), 29
 update() (frost.server.storage.base.Base class method), 27
 User (class in frost.client.objects), 17
 User (class in frost.server.database), 23
 User (class in frost.server.database.models), 22
 User (class in frost.server.storage), 29
 User (class in frost.server.storage.models), 28
 user (frost.server.database.Message attribute), 24
 user (frost.server.database.models.Message attribute), 22
 user_id (frost.server.database.Message attribute), 24

`user_id` (*frost.server.database.models.Message attribute*), [22](#)
`user_room_association` (*in module frost.server.database.models*), [23](#)
`username` (*frost.server.database.models.User attribute*), [23](#)
`username` (*frost.server.database.User attribute*), [23](#)
`UserObj` (*class in frost.server.objects*), [32](#)
`users` (*frost.server.database.models.Room attribute*), [22](#)
`users` (*frost.server.database.Room attribute*), [24](#)