PyFrost

Release 0.2.1

Dennis Pham

Apr 17, 2020

CONTENTS

PyFrost

CHAPTER

ONE

THE OFFICIAL CLIENT

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

CHAPTER

TWO

THE SOURCE CODE

The source is available here on GitHub. Feel free to contribute to this project.

CHAPTER

THREE

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() → 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]]]]]

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

Initlializes 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', MetaDat`**
An association from the many-to-many relationship between users and rooms.

Module contents

`frost.server.database.`**`init_db`**`()` → None
Initlializes 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

INDEX

E

F

G

N

O

P

R

S

T

U