# Lecture Modern Data Technologies - Exercise 1 (Redis)

**Group 6**


| Name | Matr. Nr. |
|---|---|
| David Sugar | 76050 |
| Moritz Berger | 76265 |
| Daniel Pawita | 70751 |


The source code can be found at: https://github.com/r4gus/moderne-datentechnologien/blob/master/Redis/chat.py

### Scenario

We wanna build our own small messaging app using the pubsub paradigm of Redis so we can send messages to our peers. As language for the application wee chose `Python 3` and the `redis` package. The idea is to create one channel for every user where the name of the channel is the username. The channel names are stored within a `ZSET` (sorted set) where each name is mapped to the number of unread messages. The username and a status text for each user are stored within a `HASH` where the key of each entry is `users:<uname>`. When a user sends a message, that status text is automatically appended. When a user starts the messaging client he/she gets added to a `SET` of active users. Other users can view who's currently active and start sending messages. When a user cloese the client `ctrl-C` the name of the user is removed from the set of active users. When a user receives a message it gets automatically appended to a (linked) `LIST` of received messages (history).

### Getting started

First we installed Python and Redis by using the `apt` package manager `sudo apt install python3 redis-server`. Then we created a new directory `mkdir RChat` together with a new virtual environment for Python `Python3 -m venv venv`. After that we sourced the virtual environment `source venv/bin/activate` and install the `redis` package by executing `pip install redis`.

Then we created a new python file and insert the following code.

```
import redis

if __name__ == '__main__':
    # ... implementation of the main loop
```

Here we import the downloaded `redis` package. The `__name__ == '__main__'` takes care that the following code is just executed if the file is loaded as main script.

The script expects a username and status text as command line argument when run, e.g. `(venv)$ python3 rchat.py david "hic rhodos hic salta"`.

### Implementation

A redis server connection is encapsulated into a class called `Chat` with class variables for the different data types. When a new chat is created (on program start) it tries to connect to the redis server using the given `ip` address and `port` number. This connection is stored within the `r` instance variable.

```
class Chat:
    GROUPS = 'GROUPS'
    USERS = 'users'
    ACTIVE_USERS = 'active'
    MESSAGES = 'messages'

    def __init__(self, ip, port):
        self.r = redis.Redis(host=ip, port=port, db=0)
        print(self.r)

    def set(self, key, val):
        return self.r.set(key, val)

    def get(self, key):
        return self.r.get(key)

    def get_users(self):
        """
        Get all available groups to join.
        """
        return self.r.smembers(self.ACTIVE_USERS)
```

---

A second class is used to represent a user. When a new user is created his/her `uname` and `status` message are stored within instance variables of the same

name. The user is then added to a hash set **(1)**, the number of unread messages is set to 0 **(2)**, he/she is added to the list of active users **(3)** and last but not least the user subscribes to the own channel **(4)**.

```
class User:
    """
    Each user has its own group with a name thats equal to the
    name of the user.
    """
    def __init__(self, uname, status, chat):
        self.uname = uname
        self.status = status
        self.chat = chat

        # Insert user into db                                                    # (1)
        # Hash user info        :: (HASH)
        self.chat.r.hset("{}:{}".format(chat.USERS, uname), 'uname', uname)
        self.chat.r.hset("{}:{}".format(chat.USERS, uname), 'status', status)

        # Zset of sent messages per user(group) :: (ZSET)                        # (2)
        self.chat.r.zadd(chat.GROUPS, { uname : 0.0 })

        # Add user to active users :: (SET)                                      #(3)
        self.chat.r.sadd(chat.ACTIVE_USERS, uname)

        # Enter pub/ sub and subscribe to own channel                            # (4)
        self.channel = chat.r.pubsub()
        self.channel.subscribe(uname)
```

The user name gets automatically removed from the list of active users when the associated user object is destroyed. This happens when the user closes the running app.

```
def __del__(self):
    self.chat.r.srem(chat.ACTIVE_USERS, self.uname)
```

Messages can be send to other users via the `publish` command. Before sending the message, the `send_message` method increments the amount of unread messages for the targeted user by one.

```
def send_message(self, dst, msg):
    # Increment the sent message count
    self.chat.r.zincrby(chat.GROUPS, 1.0, dst)
    return self.chat.r.publish(dst, "From {}: {}\n--------------------\n{}\n----------------
```

Another user can then check for unread messages using the `received_messages` method that returns the `ZSET` score for the given user.

```
def received_messages(self):
    return self.chat.r.zscore(chat.GROUPS, self.uname)
```

To get the pending messages one can call the `get_message` function. It fetches the next message from the subsribed channel, decodes it into `utf-8`, decreases the unread message count by one and then returns the message. The method also appends the message to a list of already received messages.

```
def get_message(self):
    msg = self.channel.get_message()
    if msg:
        message = msg['data']

        if message != None and message != 1:
            # Decode message
            msg = message.decode('utf-8')
            # Append it to the list of read messages
            self.chat.r.lpush("{}:{}".format(chat.MESSAGES, self.uname), msg)
            # Decrement count of unread messages
            self.chat.r.zincrby(chat.GROUPS, -1.0, self.uname)
            return msg
        else:
            return ""
```

If one wants to see all messages received so far one can call the `get_history` method. It gets all stored messages from the list using lrange with 0 as first index and -1 (end) as second.

```
def get_history(self):
    return self.chat.r.lrange("{}:{}".format(chat.MESSAGES, self.uname), 0, -1)
```

## Queries

1. We want to retrieve all currently active users.

```
> SMEMBERS active
1) "david"
2) "moritz"
```

2. We want to check if member 'david' is in the database.

```
> EXISTS users:david
(integer) 1
```

3. We want to get the number of 'moritz's received messages.

```
> ZSCORE GROUPS moritz
"2"
```

4. We want all list of all members and their number of received messages.

```
> ZRANGE GROUPS 0 -1 withscores
1) "david"
2) "0"
3) "moritz"
4) "2"
```

5. We want to retrieve the current state of 'moritz'.

```
> HGET users:moritz status
"Hello World"
```