HW2: Databases (Task – 2 Report)

Himanshu Gupta (UIN: 326003459)
Manish Singhal (UIN: 326007129)

Time to complete the task: Roughly 10 hours

GitHub link: https://github.tamu.edu/manish-singhal/CSCE678-HW2

Steps to run the code:

```
Instructions to run this code (Python 3 required)

Install REDIS and MONGODB

Start REDIS server:
Make sure you are in the existing path where redis-server is installed

"Himanshus-MacBook-Pro:redis-5.0.0 hg$ src/redis-server"

Start MongoDB:
for Macbook
"Himanshus-MacBook-Pro:data hg$ sudo mongod"

for windows:
"net start MongoDB"

Start the message board application:
"python3 message-boards.py"

Now following commands can be issued:

"select <MESSAGE BOARD>": This will either create a new message board or select the
already existing one

"read": Display the existing messages on the board

"write <Message>": This will send a message on the board and to all listening users

"listen": This will subscribe to the selected message board
```

<u>Working of the instructions:</u>

1) **<u>select <MESSAGE BOARD>:</u>** This will either create a new message board or select the already existing one. It looks in the collection of MongoDB for the document, if it finds one with that name, it selected for further operations otherwise a new document is created.

2) **<u>read</u>**: Display the existing messages on the selected board. It gets the content from the MongoDB for the selected board/document and displays its content.

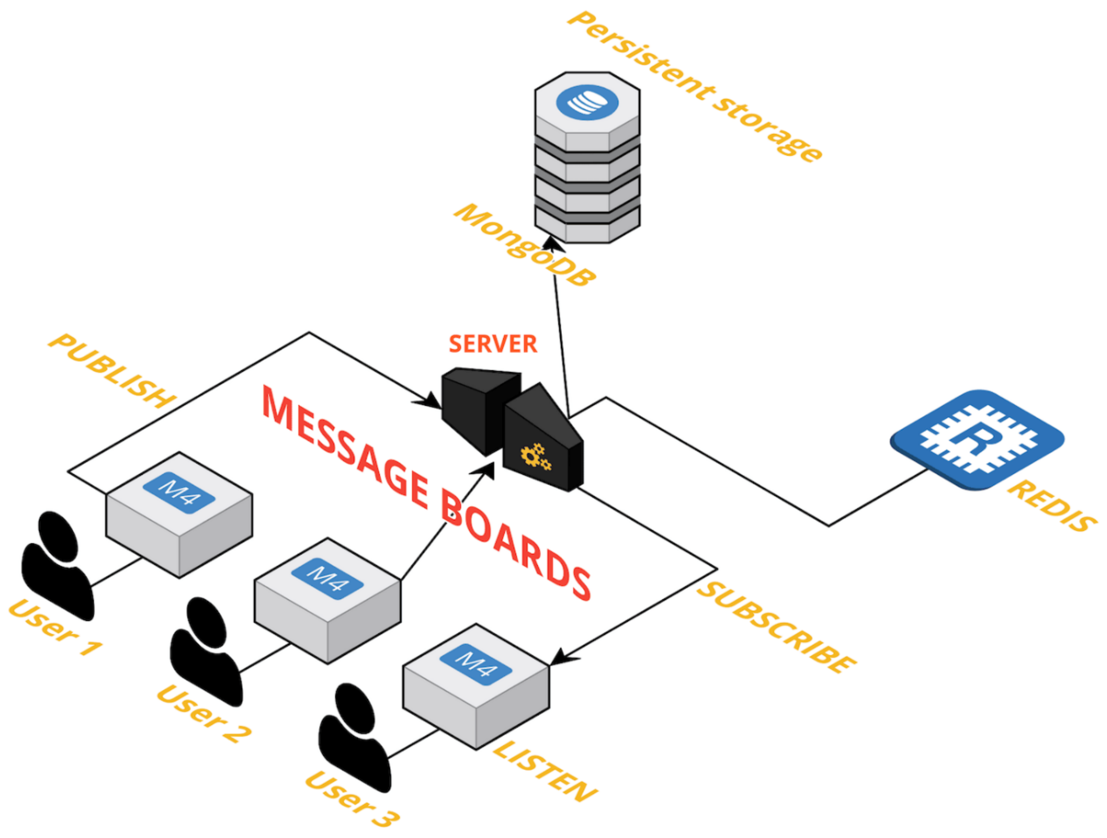
3) **<u>write <Message></u>**: This will send a message on the board and to all listening users. Also, it stores the content in MongoDB for a selected document which can be easily accessed with the help of read command later.

4) **<u>listen</u>**: This will subscribe to the selected message board. It basically uses redis feature of pubsub. When a user wants to listen, after this command user will get all updates for write operation for all users connected on message board. However, to stop listening the updates user uses Ctrl+C to generate keyboard interrupt and will be able perform read/write and other operations

5) **<u>quit</u>**: It simply return from the main program.

For the commands read, write and listen, if no board is selected, it will display the text. **<u>“Please select a message board from the list”</u>**.

System Architecture proposal:



Our model consists of a single server (running locally in our case), where all the users are connected to that server. All the users have the permission to read, write and listen once they select the proper message board from the list. The user trying to read the content will get data from the MongoDB. However, when the user wants to listen to the updates, using the feature of the pubsub user gets notified about the updates. When any user writes the data, it is stored in MongoDB.

Redis:

Redis is in-memory data structure store, used as a database and cache. Cache is helpful for faster access to the data in the future. Redis is fast and is NoSQL database.

MongoDB:

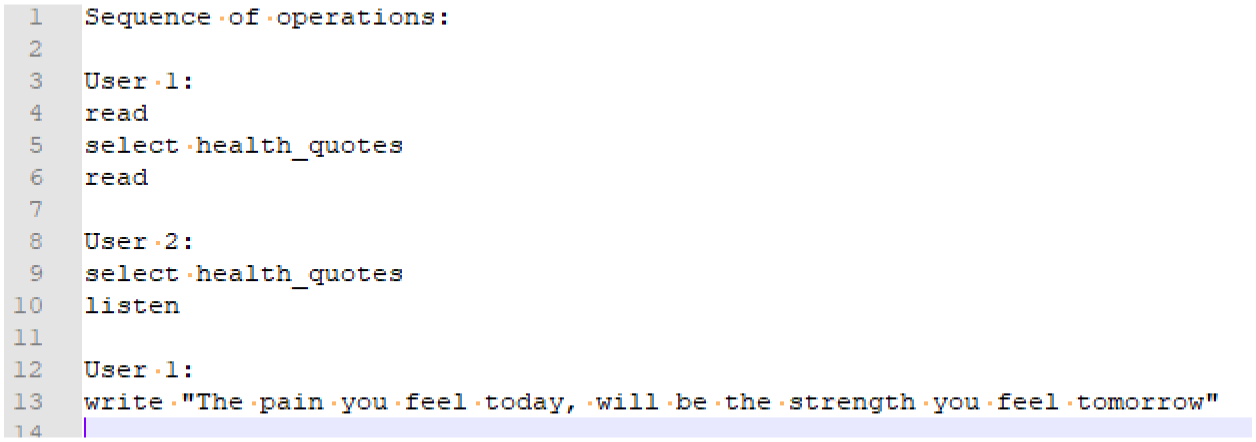
The greatest advantage of using MongoDB is the flexibility in schema design. We do not need to pre-define the fields of the data and can be modified at any point. There are many other performance benefits for using MongoDB like it can handle a very high throughput and handle more queries than a standard SQL database. It makes sure that the amount of data does not create a performance impact. The aggiefit database will have a huge amount of entries for as the users grow, the application will track their fitness history, so it can get very big with time.

MongoDB can be easily scaled up with cloud-based services. MongoDB is built to provide scalability using multiple instances and replicas. Since they do not need any schema for the data, it saves a lot of software development effort. The current data provided is a simple JSON formatted unstructured data, which can be stored as it is in the database. MongoDB uses key-value pairs to track the entries and perform search and update operations. As it does not need to support strong consistency in the data, it can be easily sharded.

<u>PubSub:</u> Publish-Subscribe is a software messaging pattern where the senders are called as publishers, don't send the message directly to its receiver called as subscribers. They are mostly implemented as a message queueing service. The subscribers will provide the class or topic on which they are interested to listen messages. Any message on that class or topic is sent to all the subscribers. Redis provides this functionality of publish and subscribe which is used in this prototype.

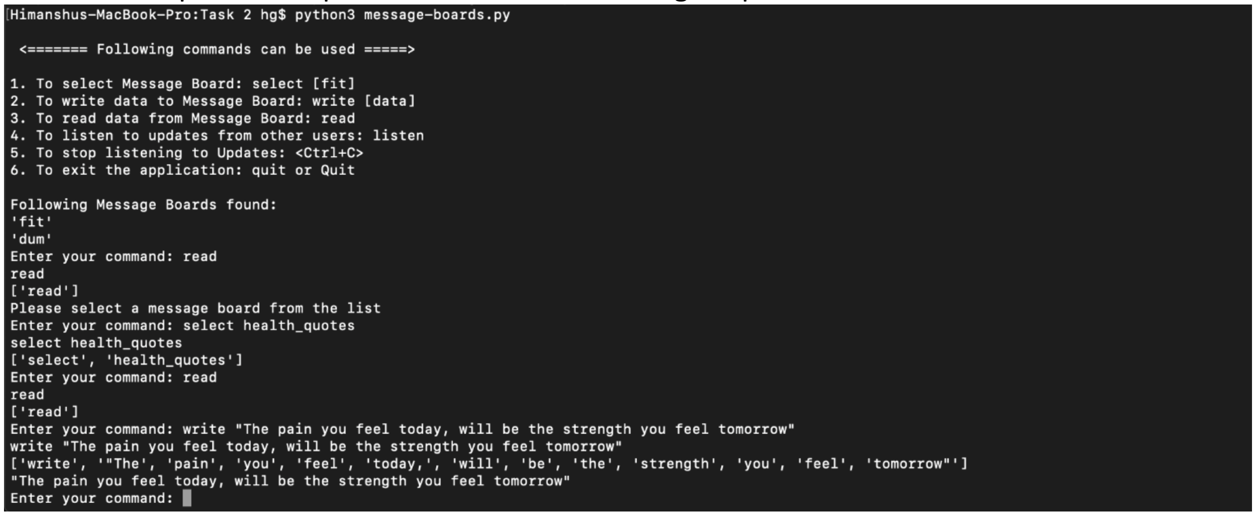
However, we cannot use Redis as a database to store the history of all the message boards as it is only an in-memory database. After reloading of any client, we will lose the data. Therefore, we will use the MongoDB database to store all the write commands.

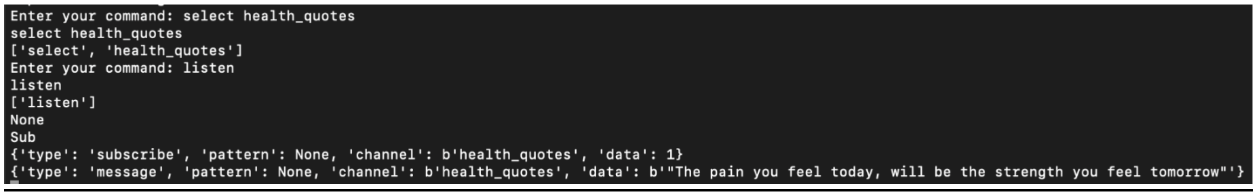
Scenario 1:



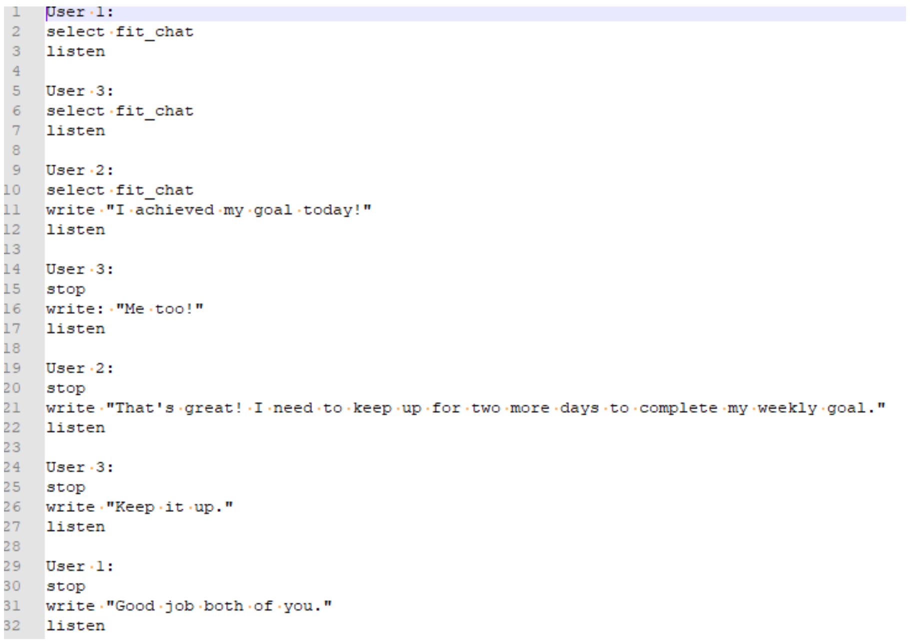
```
Sequence of operations:

User 1:
read
select health_quotes
read

User 2:
select health_quotes
listen

User 1:
write "The pain you feel today, will be the strength you feel tomorrow"

```

For above sequence of operations we have following output:

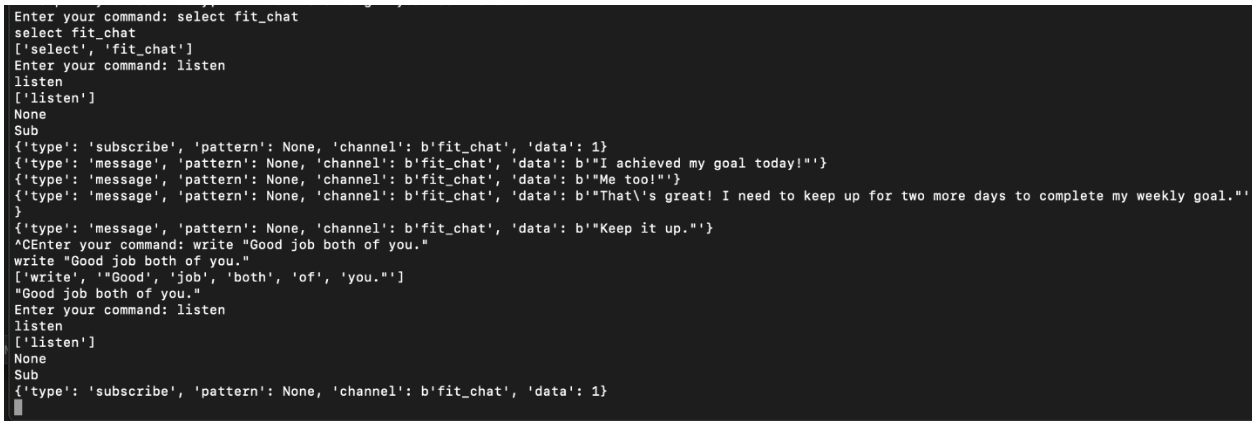

```
Himanshus-MacBook-Pro:Task 2 hg$ python3 message-boards.py

 <======= Following commands can be used =====>

1. To select Message Board: select [fit]
2. To write data to Message Board: write [data]
3. To read data from Message Board: read
4. To listen to updates from other users: listen
5. To stop listening to Updates: <Ctrl+C>
6. To exit the application: quit or Quit

Following Message Boards found:
'fit'
'dum'
Enter your command: read
read
['read']
Please select a message board from the list
Enter your command: select health_quotes
select health_quotes
['select', 'health_quotes']
Enter your command: read
read
['read']
Enter your command: write "The pain you feel today, will be the strength you feel tomorrow"
write "The pain you feel today, will be the strength you feel tomorrow"
['write', '"The', 'pain', 'you', 'feel', 'today,', 'will', 'be', 'the', 'strength', 'you', 'feel', 'tomorrow"']
"The pain you feel today, will be the strength you feel tomorrow"
Enter your command: 
```


```
Enter your command: select health_quotes
select health_quotes
['select', 'health_quotes']
Enter your command: listen
listen
['listen']
None
Sub
{'type': 'subscribe', 'pattern': None, 'channel': b'health_quotes', 'data': 1}
{'type': 'message', 'pattern': None, 'channel': b'health_quotes', 'data': b'"The pain you feel today, will be the strength you feel tomorrow"'}
```

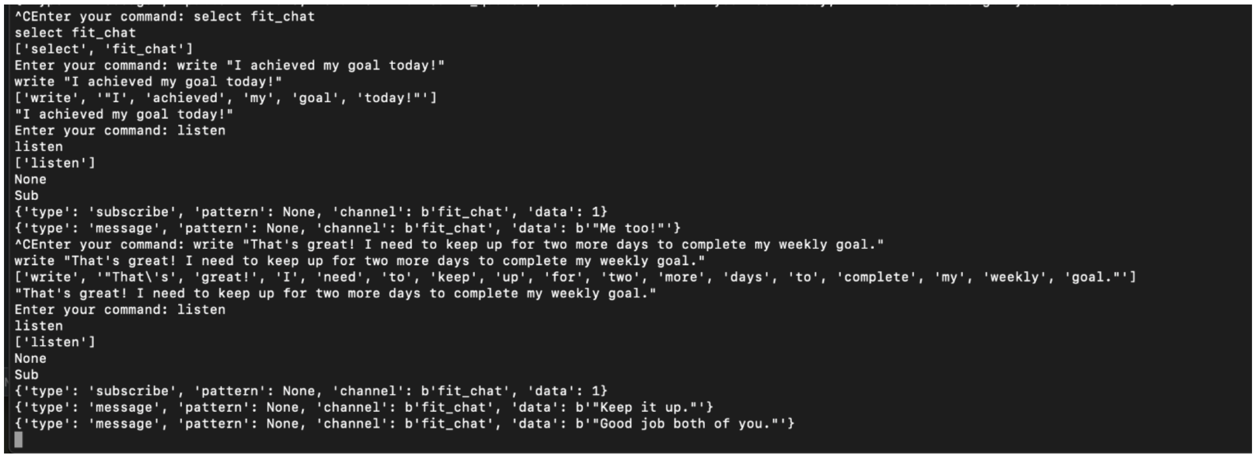
Scenario 2:


```
User 1:
select fit_chat
listen

User 3:
select fit_chat
listen

User 2:
select fit_chat
write "I achieved my goal today!"
listen

User 3:
stop
write: "Me too!"
listen

User 2:
stop
write "That's great! I need to keep up for two more days to complete my weekly goal."
listen

User 3:
stop
write "Keep it up."
listen

User 1:
stop
write "Good job both of you."
listen
```

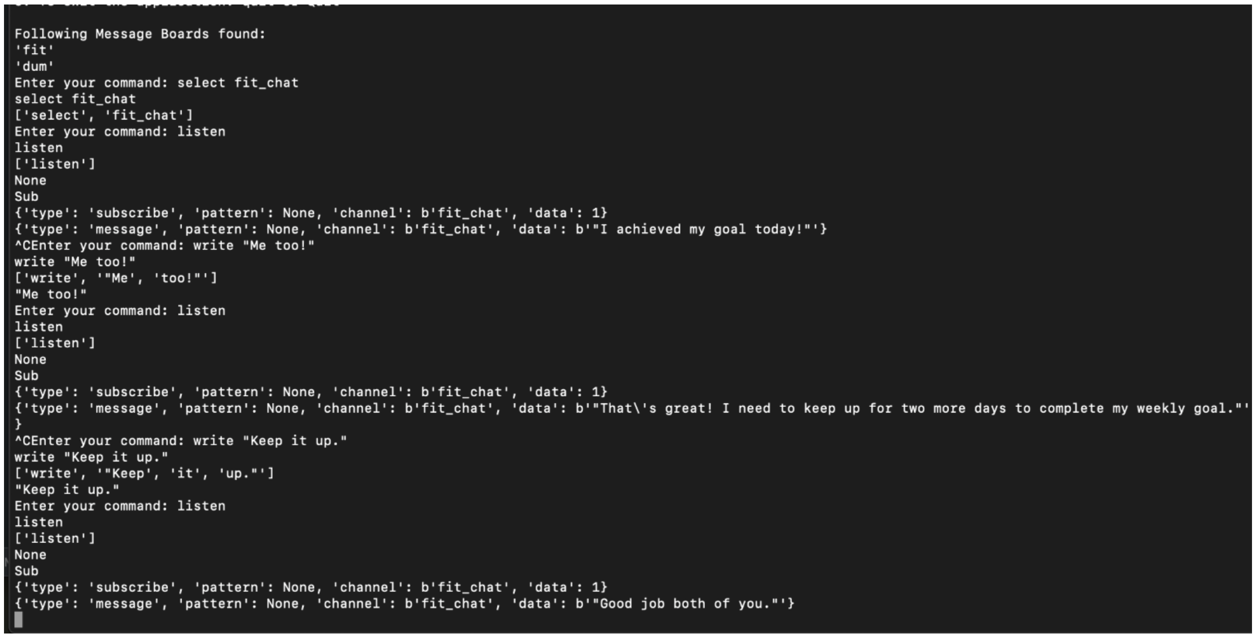
Here is the output of the above scenario:



```
Enter your command: select fit_chat
select fit_chat
['select', 'fit_chat']
Enter your command: listen
listen
['listen']
None
Sub
{'type': 'subscribe', 'pattern': None, 'channel': b'fit_chat', 'data': 1}
{'type': 'message', 'pattern': None, 'channel': b'fit_chat', 'data': b'"I achieved my goal today!"'}
{'type': 'message', 'pattern': None, 'channel': b'fit_chat', 'data': b'"Me too!"'}
{'type': 'message', 'pattern': None, 'channel': b'fit_chat', 'data': b'"That\'s great! I need to keep up for two more days to complete my weekly goal."'
}
{'type': 'message', 'pattern': None, 'channel': b'fit_chat', 'data': b'"Keep it up."'}
^CEnter your command: write "Good job both of you."
write "Good job both of you."
['write', '"Good', 'job', 'both', 'of', 'you."']
"Good job both of you."
Enter your command: listen
listen
['listen']
None
Sub
{'type': 'subscribe', 'pattern': None, 'channel': b'fit_chat', 'data': 1}
```



```
^CEnter your command: select fit_chat
select fit_chat
['select', 'fit_chat']
Enter your command: write "I achieved my goal today!"
write "I achieved my goal today!"
['write', '"I', 'achieved', 'my', 'goal', 'today!"']
"I achieved my goal today!"
Enter your command: listen
listen
['listen']
None
Sub
{'type': 'subscribe', 'pattern': None, 'channel': b'fit_chat', 'data': 1}
{'type': 'message', 'pattern': None, 'channel': b'fit_chat', 'data': b'"Me too!"'}
^CEnter your command: write "That's great! I need to keep up for two more days to complete my weekly goal."
write "That's great! I need to keep up for two more days to complete my weekly goal."
['write', '"That\'s', 'great!', 'I', 'need', 'to', 'keep', 'up', 'for', 'two', 'more', 'days', 'to', 'complete', 'my', 'weekly', 'goal."']
"That's great! I need to keep up for two more days to complete my weekly goal."
Enter your command: listen
listen
['listen']
None
Sub
{'type': 'subscribe', 'pattern': None, 'channel': b'fit_chat', 'data': 1}
{'type': 'message', 'pattern': None, 'channel': b'fit_chat', 'data': b'"Keep it up."'}
{'type': 'message', 'pattern': None, 'channel': b'fit_chat', 'data': b'"Good job both of you."'}
```



```
Following Message Boards found:
'fit'
'dum'
Enter your command: select fit_chat
select fit_chat
['select', 'fit_chat']
Enter your command: listen
listen
['listen']
None
Sub
{'type': 'subscribe', 'pattern': None, 'channel': b'fit_chat', 'data': 1}
{'type': 'message', 'pattern': None, 'channel': b'fit_chat', 'data': b'"I achieved my goal today!"'}
^CEnter your command: write "Me too!"
write "Me too!"
['write', '"Me', 'too!"']
"Me too!"
Enter your command: listen
listen
['listen']
None
Sub
{'type': 'subscribe', 'pattern': None, 'channel': b'fit_chat', 'data': 1}
{'type': 'message', 'pattern': None, 'channel': b'fit_chat', 'data': b'"That\'s great! I need to keep up for two more days to complete my weekly goal."'
}
^CEnter your command: write "Keep it up."
write "Keep it up."
['write', '"Keep', 'it', 'up."']
"Keep it up."
Enter your command: listen
listen
['listen']
None
Sub
{'type': 'subscribe', 'pattern': None, 'channel': b'fit_chat', 'data': 1}
{'type': 'message', 'pattern': None, 'channel': b'fit_chat', 'data': b'"Good job both of you."'}
```

References:

1. **https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern**
2. **https://redis.io/topics/pubsub**