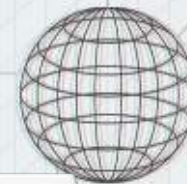




CHATTR



Hello from Chattr

ERIC

DERRICK

CONNECTED TO CHATTR

Eric joined the room

Derrick

Hey buddy!

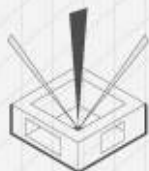
Eric

I'm having a great time over here?

Derrick joined the room

Type your message

SEND

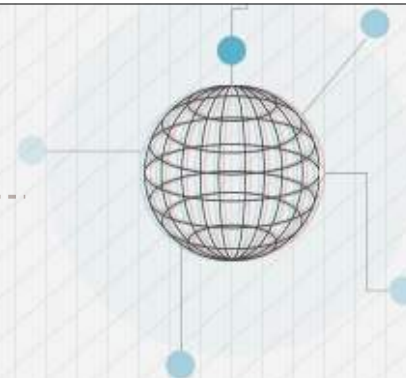


SOCKET.IO





SOCKET.IO FOR WEBSOCKETS



Abstracts websockets with fallbacks

```
$ npm install socket.io
```

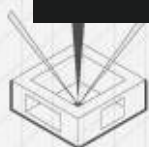
```
var socket = require('socket.io');  
var app = express.createServer();  
var io = socket.listen(app);
```

app.js

```
io.sockets.on('connection', function(client) {  
  console.log('Client connected...');  
});
```

```
<script src="/socket.io/socket.io.js"></script>  
<script>  
  var server = io.connect('http://localhost:8080');  
</script>
```

index.html



SOCKET.IO





SENDING MESSAGES TO CLIENT



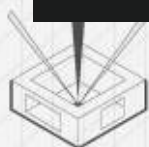
```
io.sockets.on('connection', function(client) {  
  console.log('Client connected...');  
  
  emit the 'messages' event on the client  
  client.emit('messages', { hello: 'world' });  
});
```

app.js

```
<script src="/socket.io/socket.io.js"></script>  
<script>  
  var server = io.connect('http://localhost:8080');  
  server.on('messages', function (data) {  
    alert(data.hello);  
  });  
</script>
```

index.html

listen for 'messages' events



SOCKET.IO





SENDING MESSAGES TO SERVER



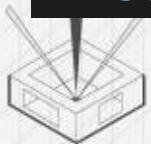
```
io.sockets.on('connection', function(client) {  
  client.on('messages', function (data) {  
    console.log(data);  
  });  
});
```

app.js

listen for 'messages' events

```
<script>  
  var server = io.connect('http://localhost:8080');  
  $('#chat_form').submit(function(e){  
    var message = $('#chat_input').val();  
    emit the 'messages' event on the server  
    socket.emit('messages', message);  
  });  
</script>
```

index.html

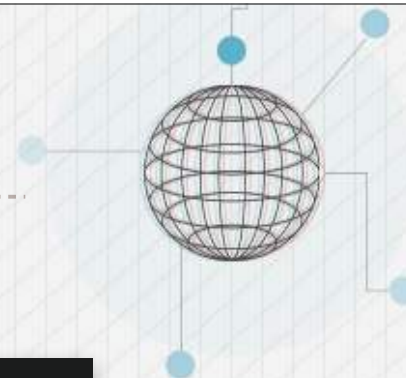


SOCKET.IO





BROADCASTING MESSAGES



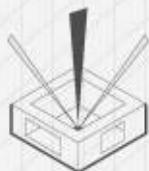
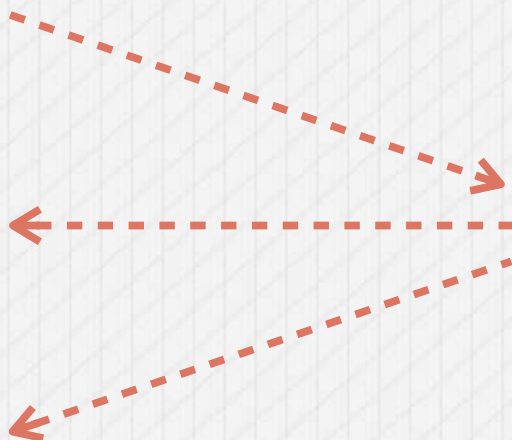
app.js

```
socket.broadcast.emit("message", 'Hello');
```

clients

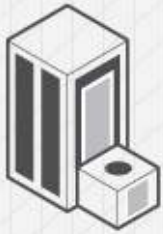


server



SOCKET.IO





BROADCASTING MESSAGES



```
io.sockets.on('connection', function(client) {  
  client.on('messages', function (data) {  
    client.broadcast.emit("messages", data);  
  });  
});
```

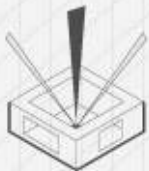
broadcast message to all other clients connected

app.js

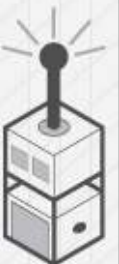
```
<script>  
  ...  
  server.on('messages', function(data) { insertMessage(data) });  
</script>
```

insert message into the chat

index.html

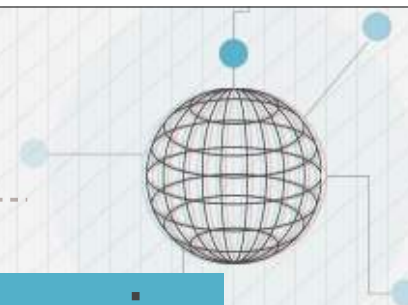


SOCKET.IO





SAVING DATA ON THE SOCKET



```
io.sockets.on('connection', function(client) {  
  client.on('join', function(name) {  
    client.set('nickname', name);  
  });  
});
```

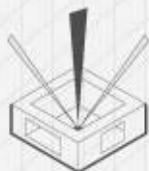
app.js

*set the nickname associated
with this client*

```
<script>  
  var server = io.connect('http://localhost:8080');  
  server.on('connect', function(data) {  
    $('#status').html('Connected to chattr');  
    nickname = prompt("What is your nickname?");  
  
    server.emit('join', nickname);  
  });  
</script>
```

index.html

*notify the server of the
users nickname*

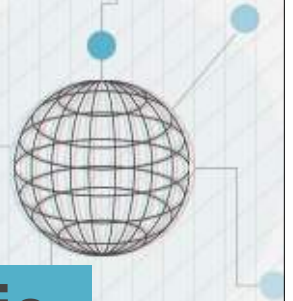


SOCKET.IO





SAVING DATA ON THE CLIENT



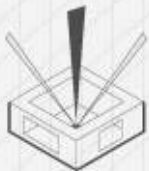
app.js

```
io.sockets.on('connection', function(client) {  
  client.on('join', function(name) {  
    client.set('nickname', name);  
  });  
  client.on('messages', function(data){  
    client.get('nickname', function(err, name) {  
      client.broadcast.emit("chat", name + ": " + message);  
    });  
  });  
});
```

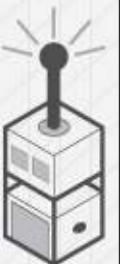
set the nickname associated with this client

get the nickname of this client before broadcasting message

broadcast with the name and message



SOCKET.IO



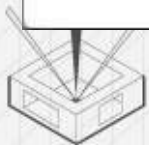


RECENT MESSAGES

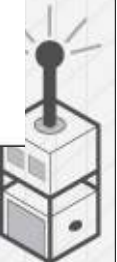


Browser window showing a chat interface:

- Tab: Hello from Chattr
- Address bar: localhost:8080
- Message area: CONNECTED TO CHATTR
- Input field: Type your message
- Send button: SEND



PERSISTING DATA



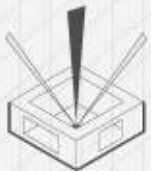


RECENT MESSAGES

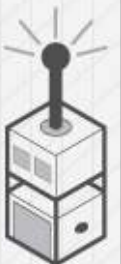


app.js

```
io.sockets.on('connection', function(client) {  
  client.on('join', function(name) {  
    client.set('nickname', name);  
    client.broadcast.emit("chat", name + " joined the chat");  
  });  
  client.on("messages", function(message){  
    client.get("nickname", function(error, name) {  
      client.broadcast.emit("messages", name + ": " + message);  
    });  
  });  
});
```



PERSISTING DATA





STORING MESSAGES



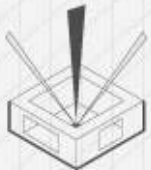
app.js

```
var messages = []; store messages in array

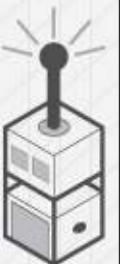
var storeMessage = function(name, data){
  messages.push({name: name, data: data}); add message to end of array
  if (messages.length > 10) {
    messages.shift(); if more than 10 messages long,
  } remove the last one
}

io.sockets.on('connection', function(client) {
  client.on("messages", function(message){
    client.get("nickname", function(error, name) {
      storeMessage(name, message);
    });
  });
});
```

*when client sends a message
call storeMessage*



PERSISTING DATA





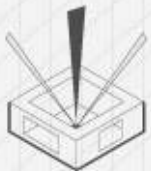
EMITTING MESSAGES



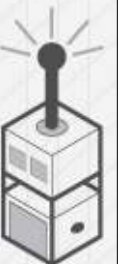
app.js

```
io.sockets.on('connection', function(client) {  
  ...  
  client.on('join', function(name) {  
    messages.forEach(function(message) {  
      client.emit("messages", message.name + ": " + message.data);  
    });  
  });  
});
```

iterate through messages array
and emit a message on the connecting client for each one



PERSISTING DATA





PERSISTING STORES



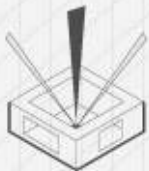
- MongoDB
- CouchDB
- PostgreSQL
- Memcached
- Riak

All non-blocking!

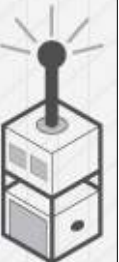


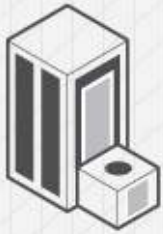
redis

Redis is a key-value store

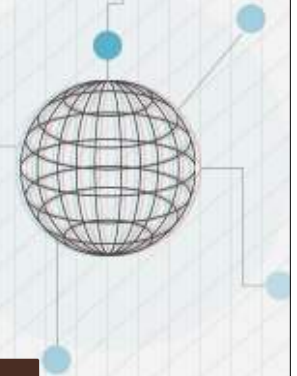


PERSISTING DATA





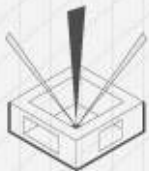
REDIS DATA STRUCTURES



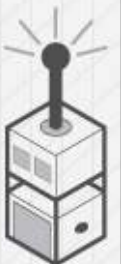
data structure

commands

Strings	SET, GET, APPEND, DECR, INCR...
Hashes	HSET, HGET, HDEL, HGETALL...
Lists	LPUSH, LREM, LTRIM, RPOP, LINSERT...
Sets	SADD, SREM, SMOVE, SMEMBERS...
Sorted Sets	ZADD, ZREM, ZSCORE, ZRANK...

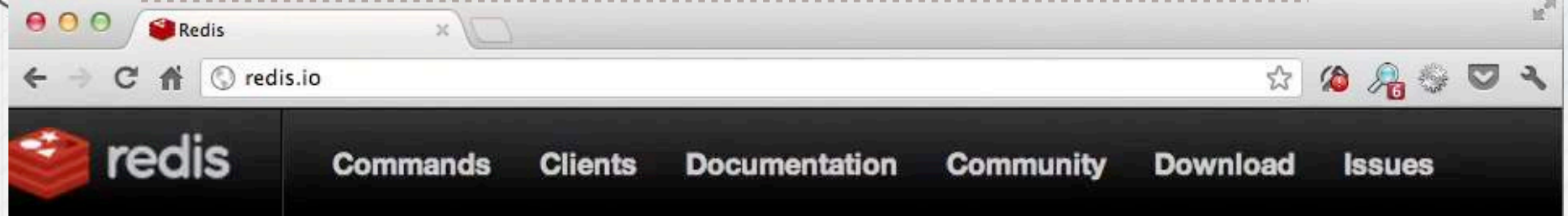


PERSISTING DATA





REDIS COMMAND DOCUMENTATION



Redis is an open source, advanced **key-value store**. It is often referred to as a **data structure server** since keys can contain [strings](#), [hashes](#), [lists](#), [sets](#) and [sorted sets](#).

[Learn more](#) →

Try it

Ready for a test drive? Check this [interactive tutorial](#) that will walk you through the most important features of Redis.

Download it

[Redis 2.4.13](#) is the latest [stable version](#). Interested in release candidates or unstable versions? [Check the downloads page](#).

What people are saying



Facebook Sets I.P.O.
Price Range
<http://t.co/7qTOhWMx>



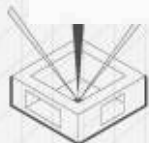
@tinkertim No more
spaces screwing my
Redis commands.
Pretty major to me ;-)



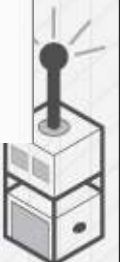
#RedMango #coupon?
Get a \$2 OFF one
@coupons.com. Just
enter your ZIP code in
the upper left! US only.
<http://t.co/P0i9nvUh>

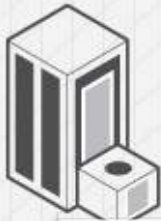


redis (@DIRTYBIITCH)

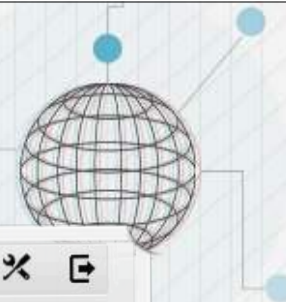


PERSISTING DATA





NODE REDIS



github

Explore Gist Blog Help



rubymaverick



😊 mranney / node_redis

Watch

Fork

1,287

109

Code

Network

Pull Requests 8

Issues 19

Wiki 1

Graphs

redis client for node — [Read more](#)

Clone in Mac

ZIP

HTTP

Git Read-Only

https://github.com/mranney/node_redis.git



Read-Only access

branch: master

Files

Commits

Branches 3

Tags 24

Downloads

Latest commit to the master branch

test.js: Switch to pubsub mode when the number of channels is > 0.



jdavisp3 authored 3 months ago

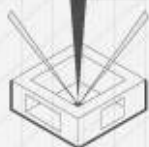
DTrejo committed 4 days ago



commit 874a893c2c

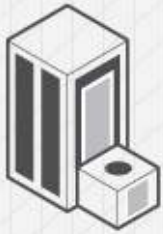
node_redis /

name	age	message	history
examples	6 months ago	Add eval example. [mranney]	
lib	6 months ago	Better util/sys fallback with try/catch instead of version magic. [mranney]	
tests	2 months ago	Add failing test. [bnoguchi]	

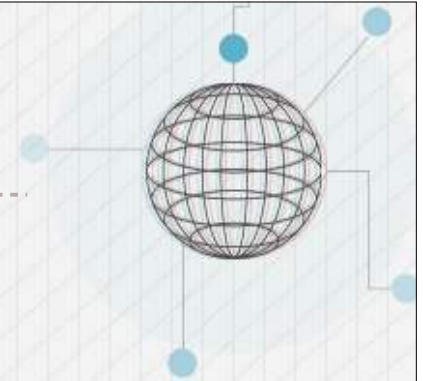


PERSISTING DATA





REDIS



```
$ npm install redis
```

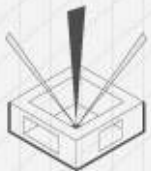
```
var redis = require('redis');  
var client = redis.createClient();  
  
client.set("message1", "hello, yes this is dog");  
client.set("message2", "hello, no this is spider");
```

key

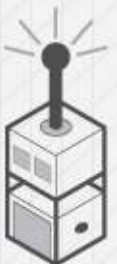
value

```
client.get("message1", function(err, reply){  
  console.log(reply); ---> "hello, yes this is dog"  
});
```

commands are non-blocking



PERSISTING DATA





REDIS LISTS: PUSHING

Add a string to the “messages” list

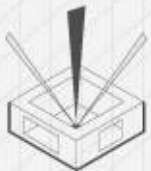
```
var message = "Hello, this is dog";  
client.lpush("messages", message, function(err, reply){  
  console.log(reply);  
});
```

→ "1" replies with list length

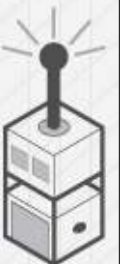
Add another string to “messages”

```
var message = "Hello, no this is spider";  
client.lpush("messages", message, function(err, reply){  
  console.log(reply);  
});
```

→ "2"



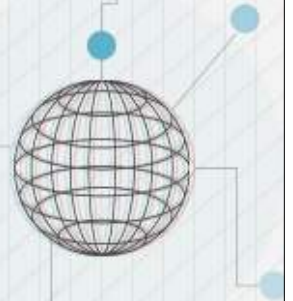
PERSISTING DATA





REDIS LISTS: RETRIEVING

Using LPUSH & LTRIM



```
var message = "Oh sorry, wrong number";  
client.lpush("messages", message, function(err, reply){  
  client.ltrim("messages", 0, 1);  
});
```

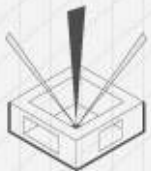
*trim keeps first two strings
and removes the rest*

Retrieving from list

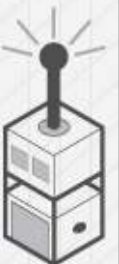
```
client.lrange("messages", 0, -1, function(err, messages){  
  console.log(messages);  
})
```

replies with all strings in list

`["Hello, no this is spider", "Oh sorry, wrong number"]`

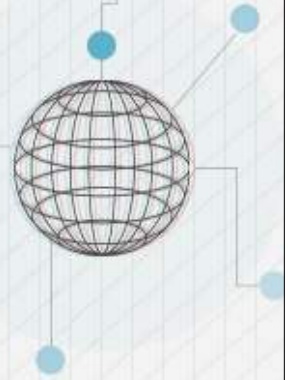


PERSISTING DATA





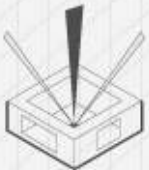
CONVERTING MESSAGES TO REDIS



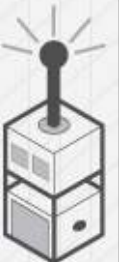
```
var storeMessage = function(name, data){  
  messages.push({name: name, data: data});  
  
  if (messages.length > 10) {  
    messages.shift();  
  }  
}
```

app.js

Let's use the List data-structure

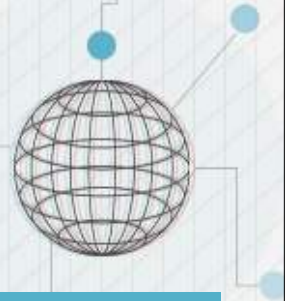


PERSISTING DATA





CONVERTING STOREMESSAGE



app.js

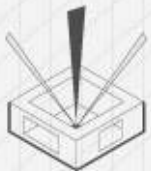
```
var redisClient = redis.createClient();
```

```
var storeMessage = function(name, data){  
  var message = JSON.stringify({name: name, data: data});
```

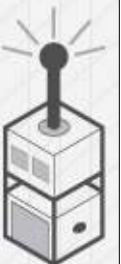
need to turn object into string to store in redis

```
  redisClient.lpush("messages", message, function(err, response) {  
    redisClient.ltrim("messages", 0, 9);  
  });  
}
```

keeps newest 10 items

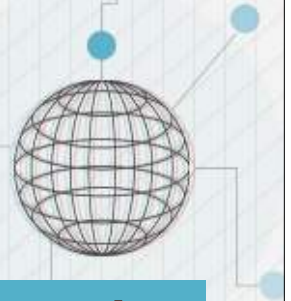


PERSISTING DATA



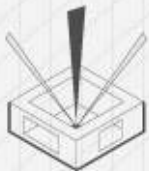


OUTPUT FROM LIST

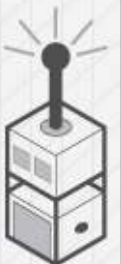


```
client.on('join', function(name) {  
  messages.forEach(function(message) {  
    client.emit("messages", message.name + ": " + message.data);  
  });  
});
```

app.js

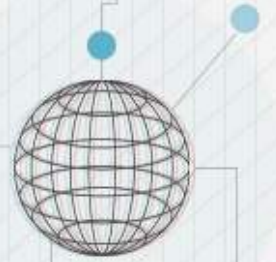


PERSISTING DATA





OUTPUT FROM LIST

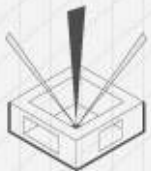


app.js

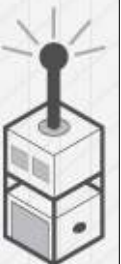
```
client.on('join', function(name) {  
  redisClient.lrange("messages", 0, -1, function(err, messages){  
    messages = messages.reverse();  
  
    messages.forEach(function(message) {  
      message = JSON.parse(message);  
      client.emit("messages", message.name + ": " + message.data);  
    });  
  });  
});
```

reverse so they are emitted in correct order

parse into JSON object



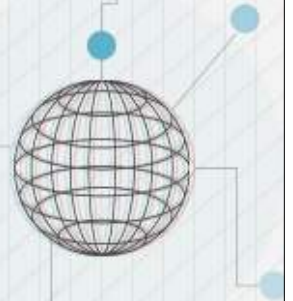
PERSISTING DATA





CURRENT CHATTER LIST

Sets are lists of unique data



DOG
SPIDER
GREGG

add & remove members of the names set

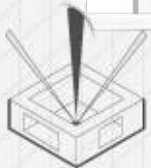
```
client.sadd("names", "Dog");  
client.sadd("names", "Spider");  
client.sadd("names", "Gregg");
```

```
client.srem("names", "Spider");
```

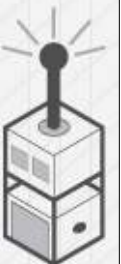
reply with all members of set

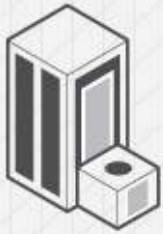
```
client.smembers("names", function(err, names){  
  console.log(names);  
});
```

["Dog", "Gregg"]

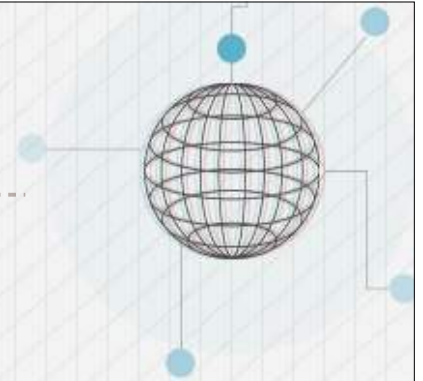


PERSISTING DATA





ADDING CHATTERS



```
client.on('join', function(name){
```

app.js

notify other clients a chatter has joined

```
client.broadcast.emit("add chatter", name);
```

```
redisClient.sadd("chatters", name);
```

```
});
```

add name to chatters set

```
server.on('add chatter', insertChatter);
```

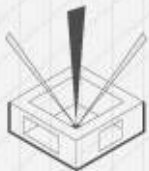
index.html

```
var insertChatter = function(name) {
```

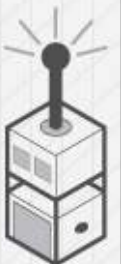
```
  var chatter = $('<li>'+name+'</li>').data('name', name);
```

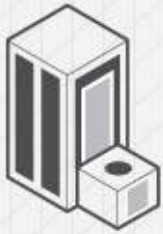
```
  $('#chatters').append(chatter);
```

```
}
```

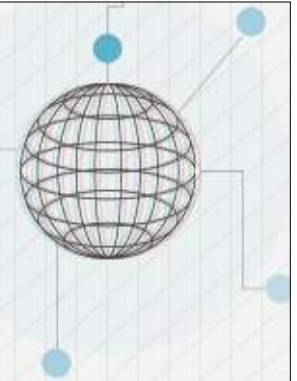


PERSISTING DATA



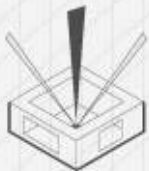


ADDING CHATTERS (CONT)



app.js

```
client.on('join', function(name){  
  notify other clients a chatter has joined  
  client.broadcast.emit("add chatter", name);  
  redisClient.smembers('names', function(err, names) {  
    names.forEach(function(name){  
      client.emit('add chatter', name);  
    });  
  });  
  emit all the currently logged in chatters  
  to the newly connected client  
});  
  
redisClient.sadd("chatters", name);  
});  
add name to chatters set
```

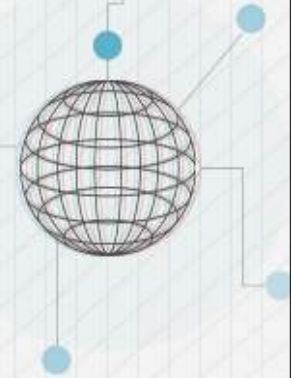


PERSISTING DATA





REMOVING CHATTERS



remove chatter when they disconnect from server

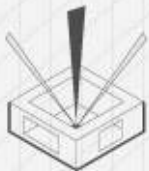
```
client.on('disconnect', function(name){  
  client.get('nickname', function(err, name){  
    client.broadcast.emit("remove chatter", name);  
    redisClient.srem("chatters", name);  
  });  
});
```

app.js

```
server.on('remove chatter', removeChatter);
```

index.html

```
var removeChatter = function(name) {  
  $('#chatters li[data-name=' + name + ']').remove();  
}
```



PERSISTING DATA

