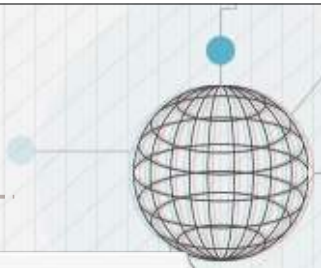
CHATTR



Hello from Chattr

ERIC

DERRICK

CONNECTED TO CHATTR

Eric *joined the room*

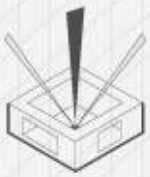
Derrick
Hey buddy!

Eric
I'm having a great time over here?

Derrick *joined the room*

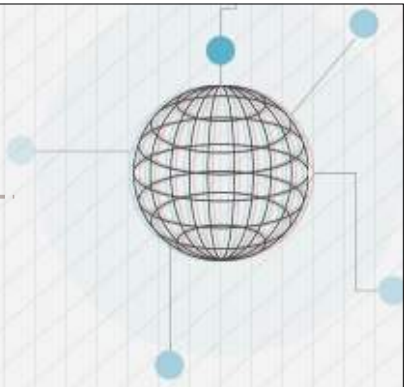
Type your message

SEND



SOCKET.IO

SOCKET.IO FOR WEBSOCKETS



Abstracts websockets with fallbacks

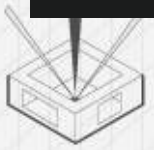
```
$ npm install socket.io
```

app.js

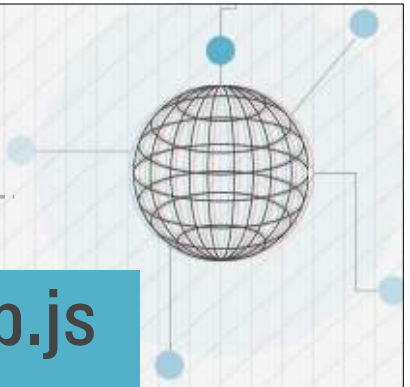
```
var socket = require('socket.io');
var app = express.createServer();
var io = socket.listen(app);


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

index.html

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



SENDING MESSAGES TO CLIENT



app.js

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

    emit the 'messages' event on the client
  client.emit('messages', { hello: 'world' });
});
```

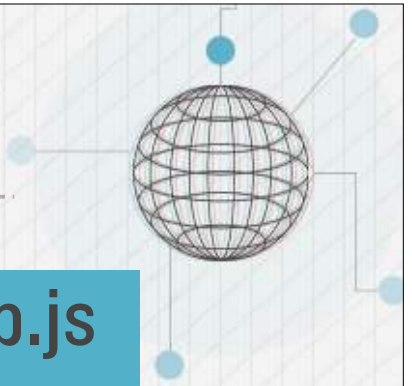
index.html

```
<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>
```

listen for 'messages' events

SOCKET.IO

SENDING MESSAGES TO SERVER



app.js

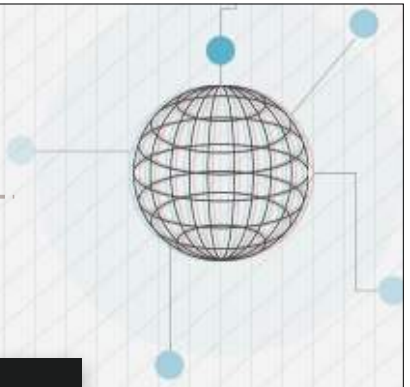
```
io.sockets.on('connection', function(client) {
  client.on('messages', function (data) {
    console.log(data);
  });
});
```

listen for 'messages' events

index.html

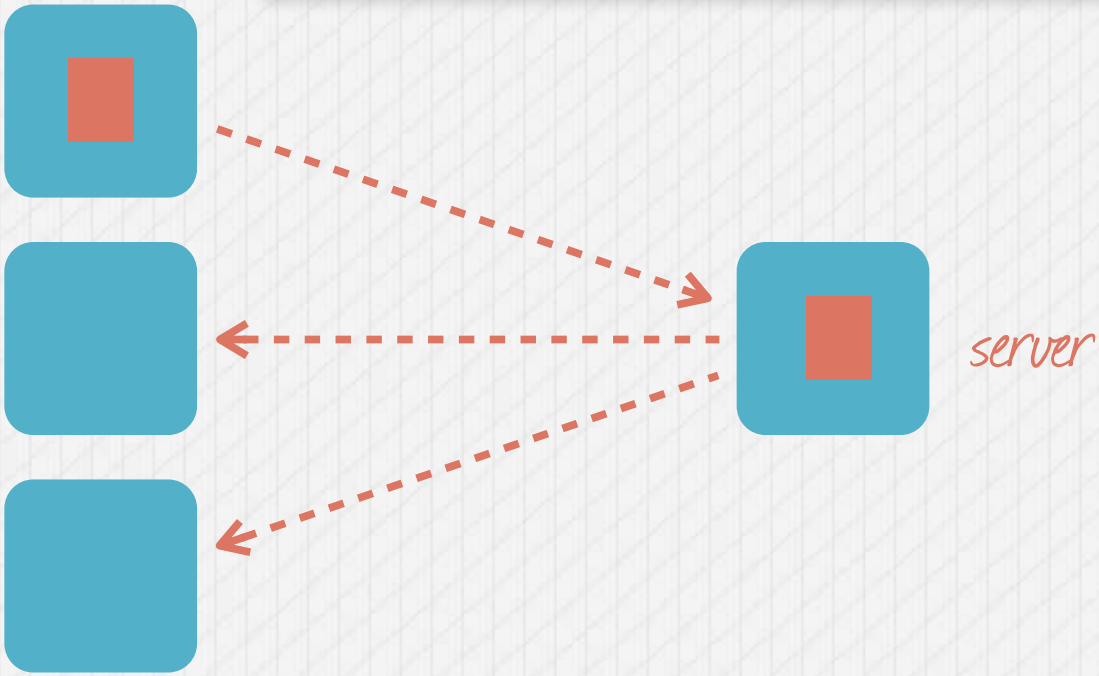
```
<script>
  var server = io.connect('http://localhost:8080');
  $('#chat_form').submit(function(e){
    var message = $('#chat_input').val();
    emit the 'messages' event on the server
    socket.emit('messages', message);
  });
</script>
```

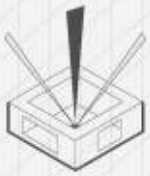
BROADCASTING MESSAGES



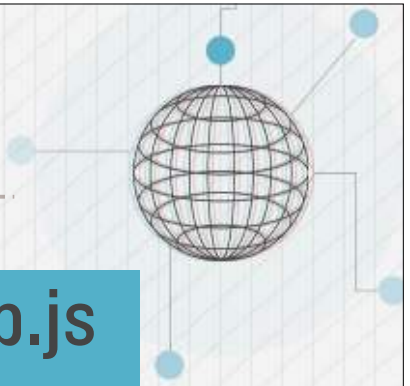
app.js

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





BROADCASTING MESSAGES



app.js

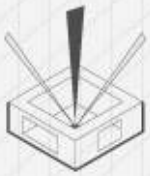
```
io.sockets.on('connection', function(client) {
  client.on('messages', function (data) {
    client.broadcast.emit("messages", data);
  });
});
```

broadcast message to all other clients connected

index.html

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

insert message into the chat



SAVING DATA ON THE SOCKET

app.js

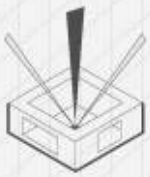
```
io.sockets.on('connection', function(client) {
  client.on('join', function(name) {
    client.set('nickname', name);
  });
});
```

set the nickname associated with this client

index.html

```
<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>
```

notify the server of the users nickname



SAVING DATA ON THE CLIENT

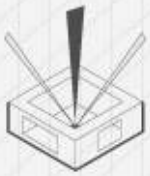
app.js

```
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);
    });
  });
});
```

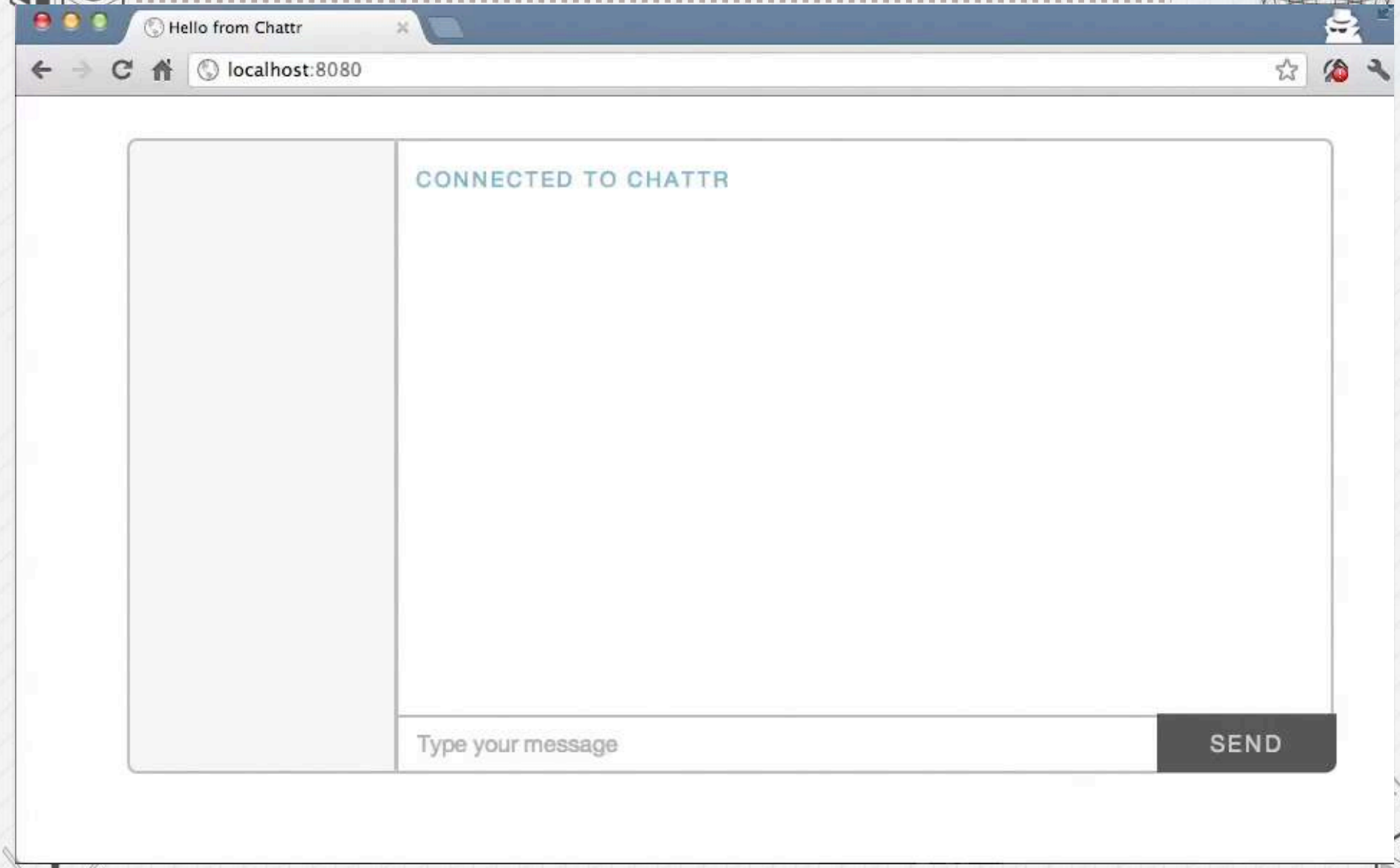
set the nickname associated with this client

get the nickname of this client before broadcasting message

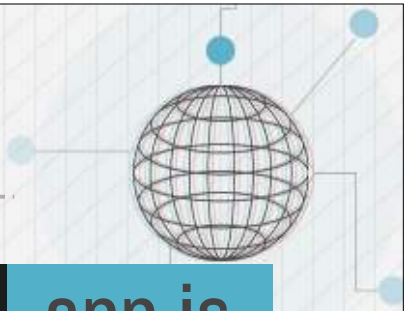
broadcast with the name and message



RECENT MESSAGES

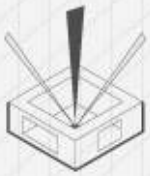


RECENT MESSAGES

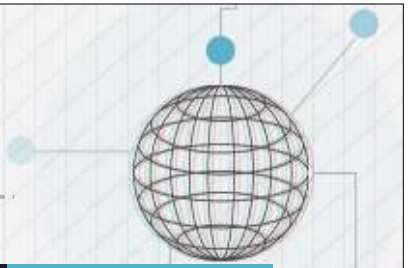


app.js

```
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);
    });
  });
});
```

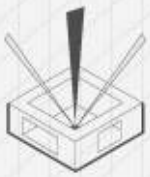


STORING MESSAGES

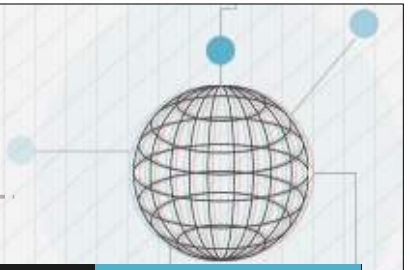


app.js

```
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
});
```



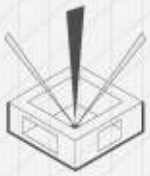
EMITTING MESSAGES



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

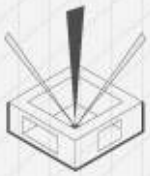
PERSISTING STORES

- MongoDB
- CouchDB
- PostgreSQL
- Memcached
- Riak

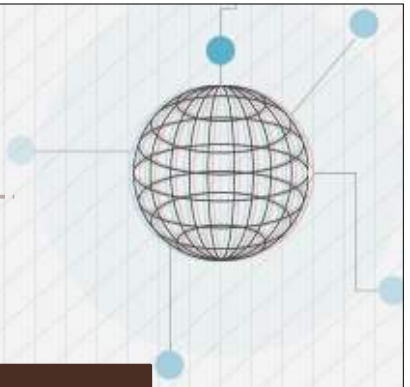
All non-blocking!

redis

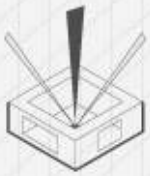
Redis is a key-value store



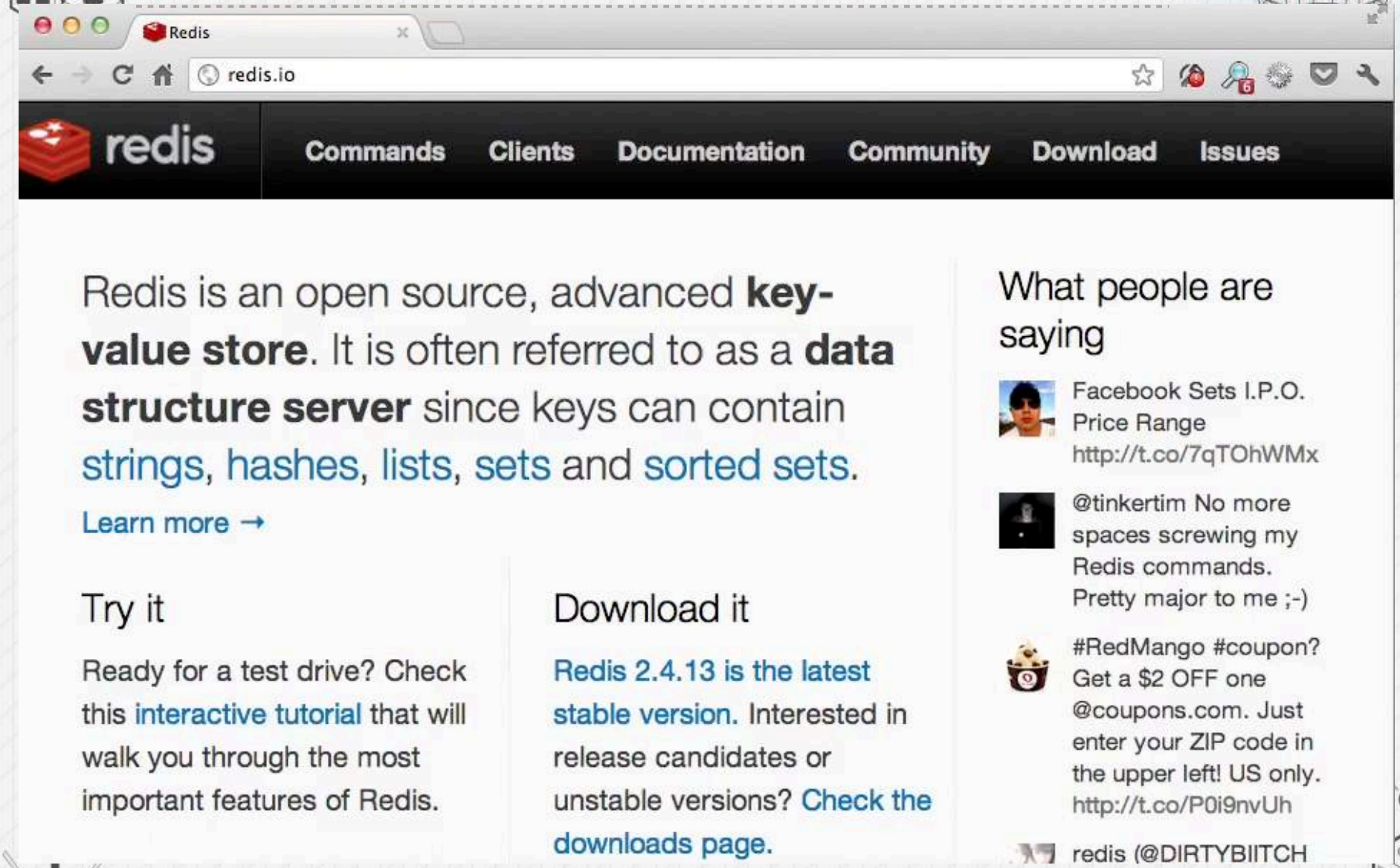
REDIS DATA STRUCTURES



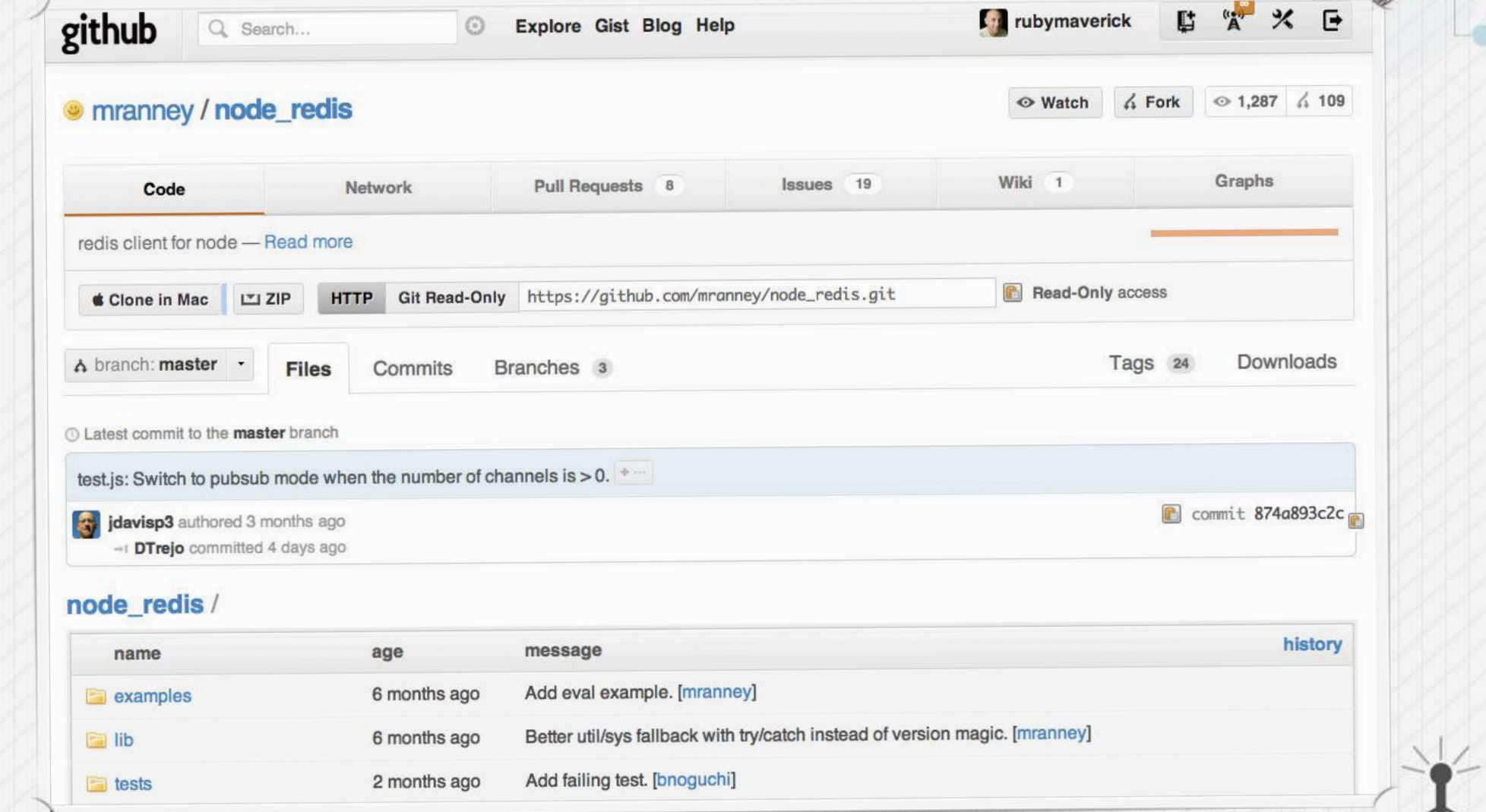
data structure	commands
Strings	SET, GET, APPEND, DECR, INCR...
Hashes	HSET, HGET, HDEL, HGETALL...
Lists	LPUSH, LREM, LTRIM, RPOP, LINSERT...
Sets	SADD, SREM, SMOVE, SMEMBERS...
Sorted Sets	ZADD, ZREM, ZSCORE, ZRANK...



REDIS COMMAND DOCUMENTATION

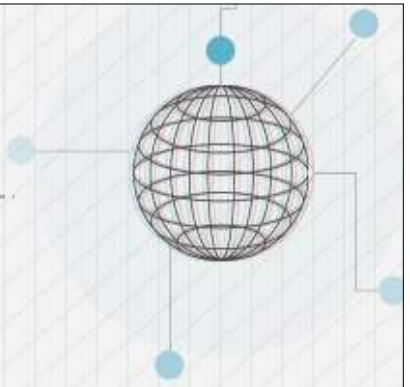


NODE REDIS



PERSISTING DATA

REDIS



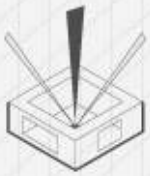
```
$ npm install redis
```

```
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

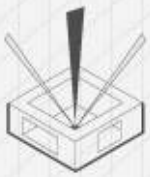
REDIS LISTS: PUSHING

Add a string to the “messages” list

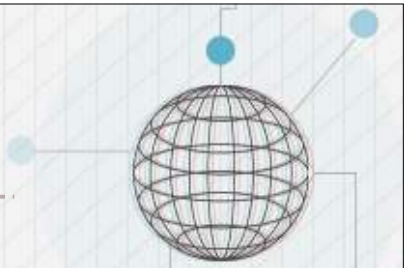
```
var message = "Hello, this is dog";
client.lpush("messages", message, function(err, reply){
  console.log(reply); ----> "1"   replies with list length
});
```

Add another string to “messages”

```
var message = "Hello, no this is spider";
client.lpush("messages", message, function(err, reply){
  console.log(reply); ----> "2"
});
```



REDIS LISTS: RETRIEVING



Using LPUSH & LTRIM

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

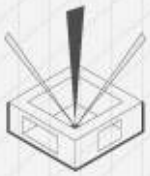
trim keeps first two strings and removes the rest

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"]
```

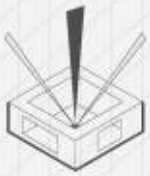


CONVERTING MESSAGES TO REDIS

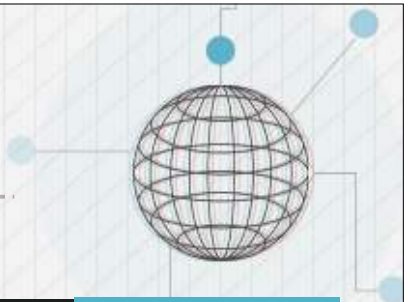
app.js

```
var storeMessage = function(name, data){
  messages.push({name: name, data: data});

  if (messages.length > 10) {
    messages.shift();
  }
}
```

Let's use the List data-structure



CONVERTING STOREMESSAGE

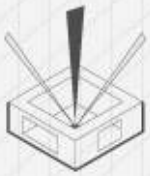


app.js

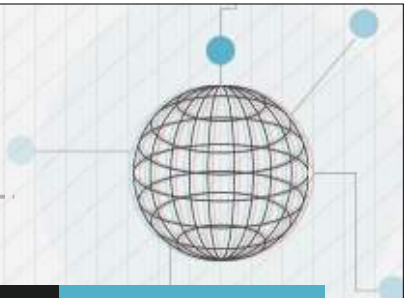
```
var redisClient = redis.createClient();

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

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

need to turn object into string to store in redis

keeps newest 10 items

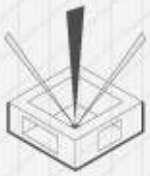


OUTPUT FROM LIST

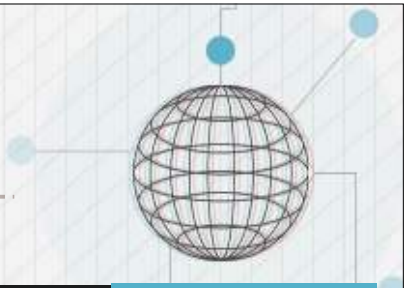


app.js

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



OUTPUT FROM LIST

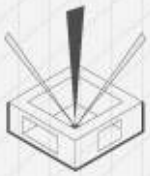


app.js

```
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



CURRENT CHATTER LIST

Sets are lists of unique data

add & remove members of the names set

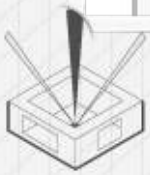
```
client.sadd("names", "Dog");
client.sadd("names", "Spider");
client.sadd("names", "Gregg");
```

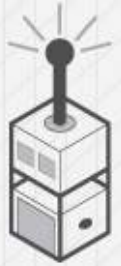
```
client.srem("names", "Spider");
```

reply with all members of set

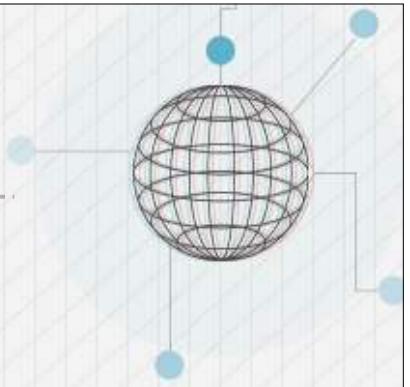
```
client.smembers("names", function(err, names){
  console.log(names);
});
```

```
["Dog", "Gregg"]
```





ADDING CHATTERS

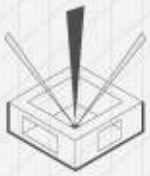


app.js

```
client.on('join', function(name){
  notify other clients a chatter has joined
  client.broadcast.emit("add chatter", name);
  redisClient.sadd("chatters", name);
});
        add name to chatters set
```

index.html

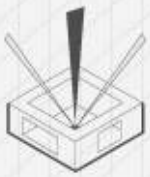
```
server.on('add chatter', insertChatter);
var insertChatter = function(name) {
  var chatter = $('<li>'+name+'</li>').data('name', name);
  $('#chatters').append(chatter);
}
```



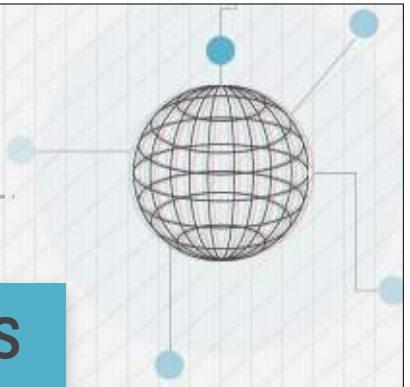
ADDING CHATTERS (CONT)

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
```



REMOVING CHATTERS



remove chatter when they disconnect from server

app.js

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

    redisClient.srem("chatters", name);
  });
});
```

index.html

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

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