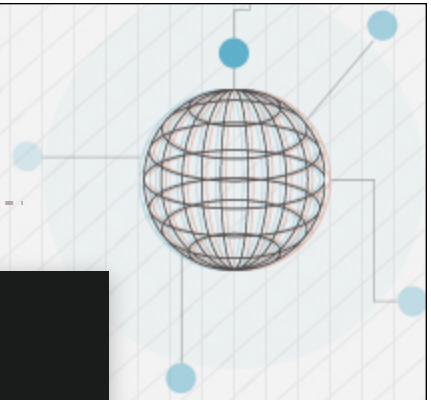
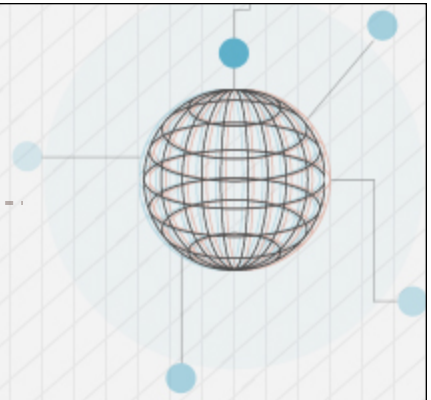
LETS CREATE AN ECHO SERVER



```
http.createServer(function(request, response) {
  response.writeHead(200);
  request.on('data', function(chunk) {
    response.write(chunk);
  });

  request.on('end', function() {
    response.end();
  });
}).listen(8080)
```

```
request.pipe(response);
```

LETS CREATE AN ECHO SERVER!



```
http.createServer(function(request, response) {
  response.writeHead(200);
  request.pipe(response);
}).listen(8080)
```

```
$ curl -d 'hello' http://localhost:8080
```

----> Hello *on client*

Kinda like on the command line

```
cat 'bleh.txt' | grep 'something'
```

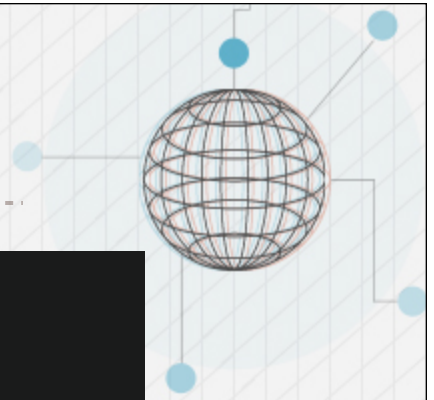
READING AND WRITING A FILE

```
var fs = require('fs');
```

require filesystem module

```
var file = fs.createReadStream("readme.md");
var newFile = fs.createWriteStream("readme_copy.md");

file.pipe(newFile);
```

UPLOAD A FILE



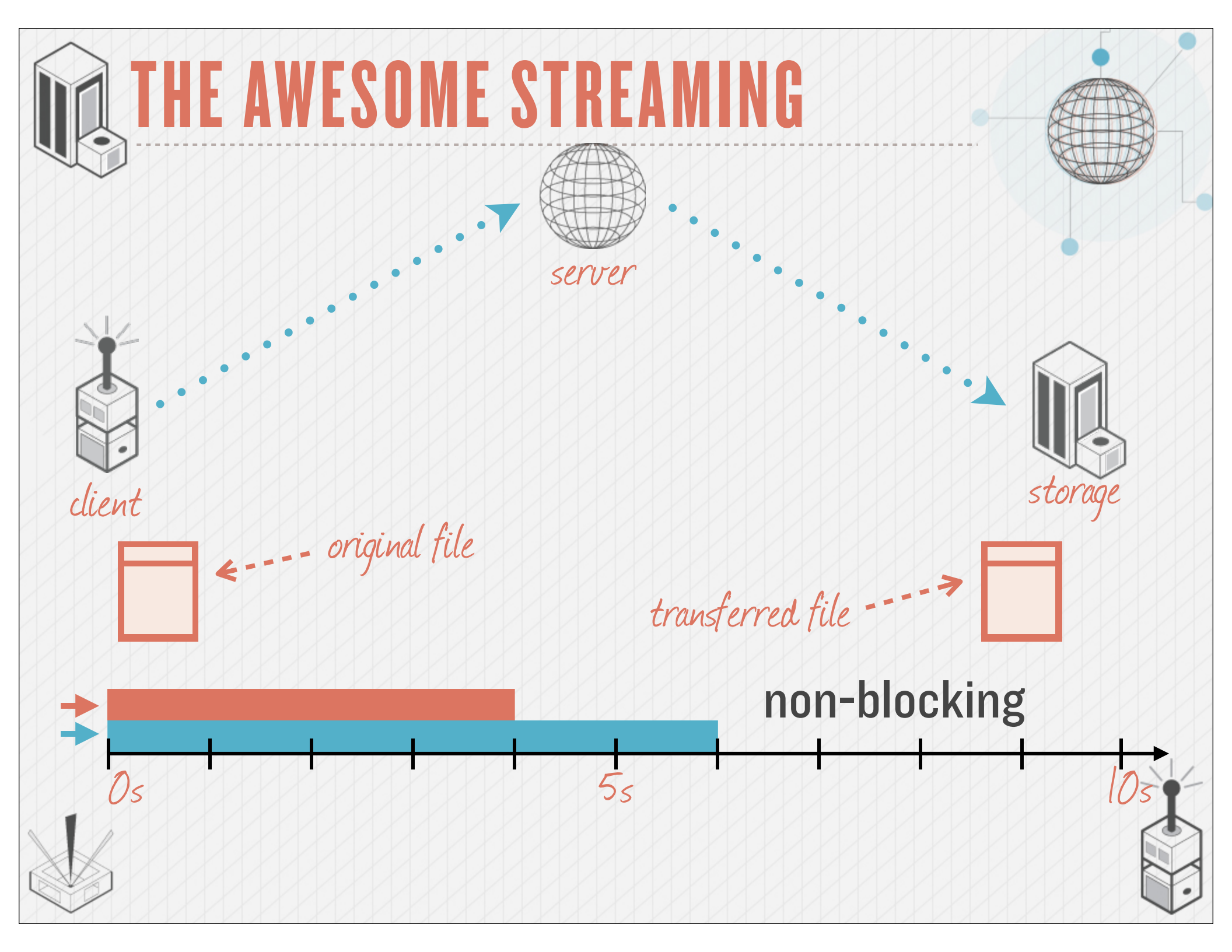
```
var fs = require('fs');
var http = require('http');

http.createServer(function(request, response) {
   var newFile = fs.createWriteStream("readme_copy.md");
   request.pipe(newFile);

   request.on('end', function() {
     response.end('uploaded!');
   });
}).listen(8080);
```

```
$ curl --upload-file readme.md http://localhost:8080
```

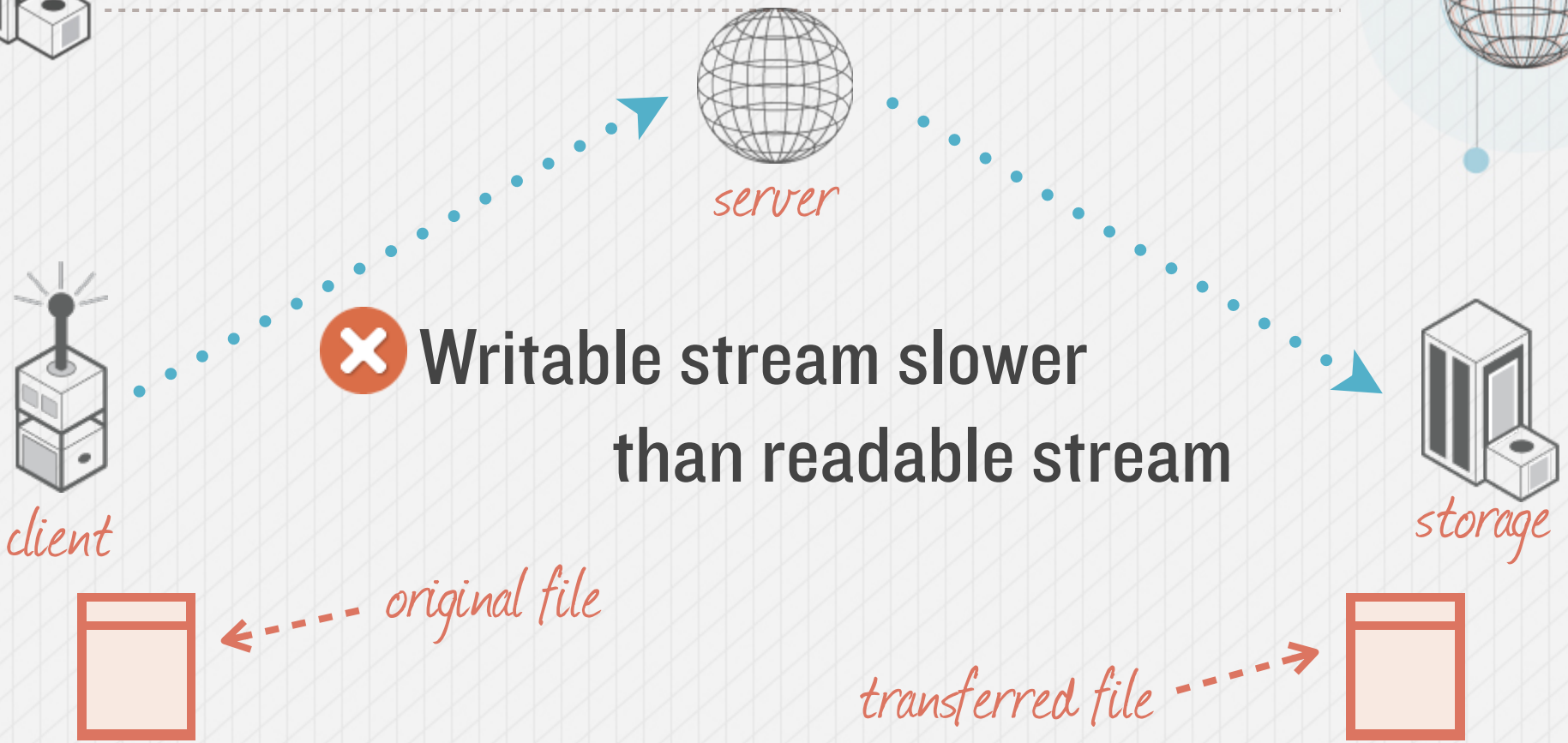
----> uploaded!

STREAMS

THE AWESOME STREAMING



BACK PRESSURE!

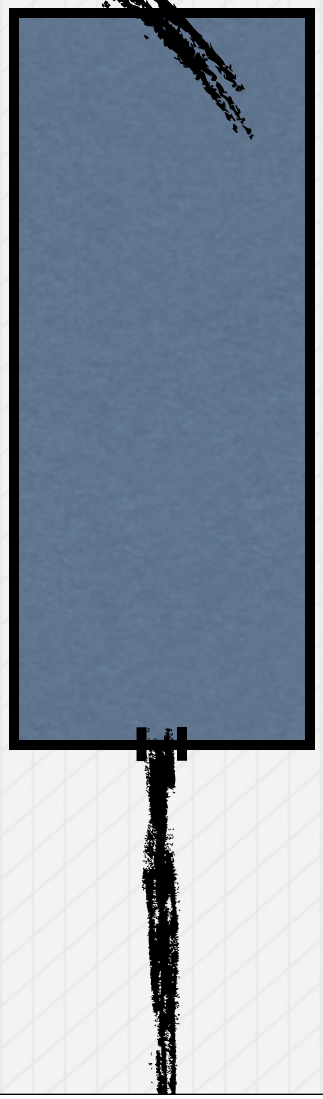


Writable stream slower than readable stream

Using pipe solves this problem

THINK OF A MILK JUG

```
milkStream.pause();
```



```
Once milk jug is drained
    milkStream.resume();
```

PIPE SOLVES BACKPRESSURE

Pause when writeStream is full

```
readStream.on('data', function(chunk) {
  var buffer_good = writeStream.write(chunk);
  if (!buffer_good) readStream.pause();
});
```

returns false if kernel buffer full

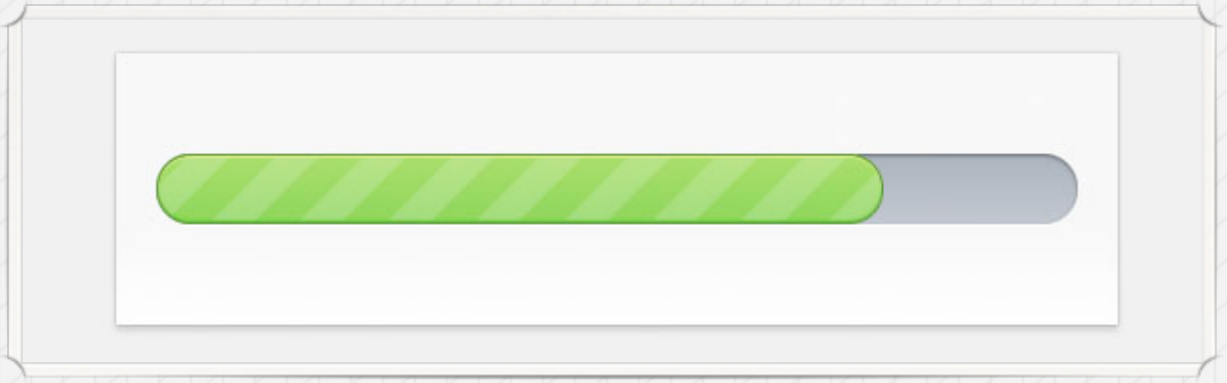
Resume when ready to write again

```
writeStream.on('drain', function(){
  readStream.resume();
});
```

All encapsulated in

```
readStream.pipe(writeStream);
```

FILE UPLOADING PROGRESS

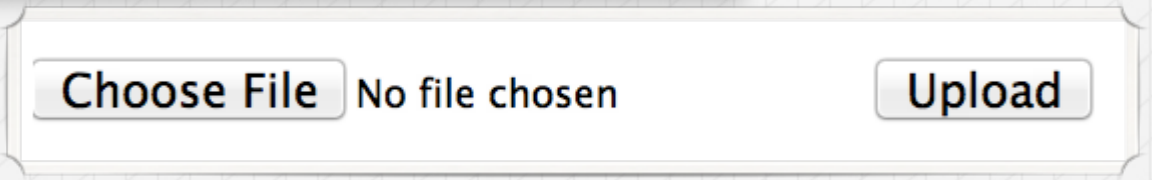


FILE UPLOADING PROGRESS

```
$ curl --upload-file file.jpg http://localhost:8080
```

Outputs:

```
progress: 3%
progress: 6%
progress: 9%
progress: 12%
progress: 13%
...
progress: 99%
progress: 100%
```



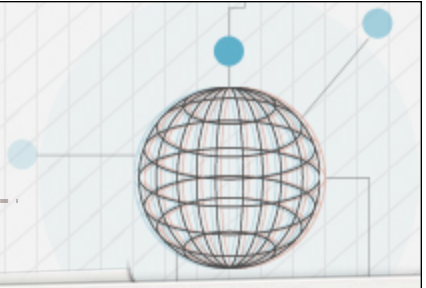
We're going to need:

- HTTP Server
- File System

STREAMS

DOCUMENTATION

http://nodejs.org/api/



Stability Scores

File System

```
Stability: 3 - Stable
```

File I/O is provided by simple wrappers around standard PO
`require('fs')`. All the methods have asynchronous and sy

The asynchronous form always take a completion callback as
completion callback depend on the method, but the first arg
operation was completed successfully, then the first argumen

When using the synchronous form any exceptions are imme
exceptions or allow them to bubble up.

Here is an example of the asynchronous version:

```
var fs = require('fs');

fs.unlink('/tmp/hello', function (err) {
```

Stream

```
Stability: 2 - Unstable
```

A stream is an abstract interface implemented by various objects in Node
server is a stream, as is stdout. Streams are readable, writable, or both. Al
`EventEmitter`.

You can load up the Stream base class by doing `require('stream')`.

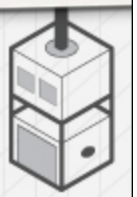
Readable Stream

A `Readable Stream` has the following methods, members, and events.

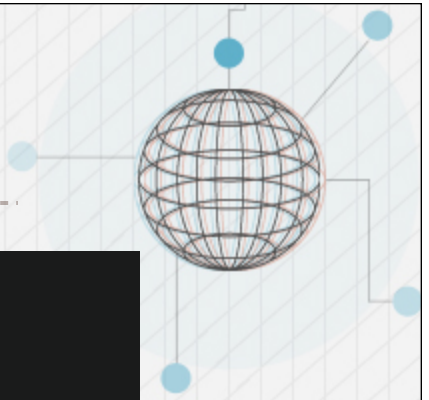
Event: 'data'

`function (data) { }`

The `'data'` event emits either a `Buffer` (by default) or a string if `set`



REMEMBER THIS CODE?

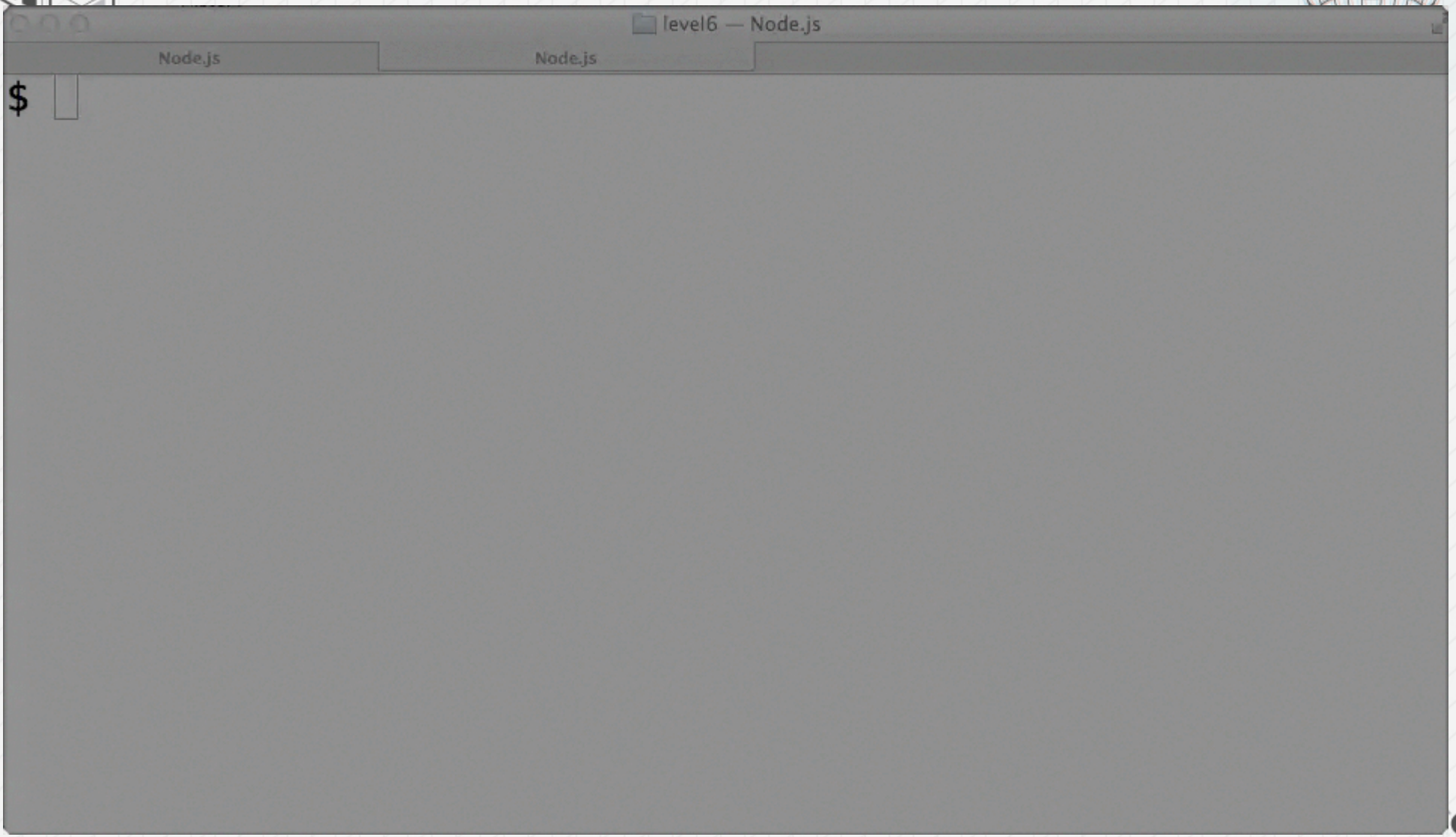


```
var fs = require('fs');
var http = require('http');

http.createServer(function(request, response) {
   var newFile = fs.createWriteStream("readme_copy.md");
   request.pipe(newFile);

   request.on('end', function() {
     response.end('uploaded!');
   });
}).listen(8080);
```

REMEMBER THIS CODE?

```
http.createServer(function(request, response) {
   var newFile = fs.createWriteStream("readme_copy.md");
   var fileBytes = request.headers['content-length'];
   var uploadedBytes = 0;

   request.pipe(newFile);

   request.on('data', function(chunk) {
     uploadedBytes += chunk.length;
     var progress = (uploadedBytes / fileBytes) * 100;
     response.write("progress: " + parseInt(progress, 10) + "%\n");
   });
   ...
}).listen(8080);
```

SHOWING PROGRESS



MODULES

- LEVEL FOUR -

REQUIRING MODULES

```
var http = require('http');
```

--------> http.js

```
var fs = require('fs');
```

--------> fs.js

How does ‘require’ return the libraries?

How does it find these files?

LETS CREATE OUR OWN MODULE

custom_hello.js

```
var hello = function() {
  console.log("hello!");
}

exports = hello;
```

custom_goodbye.js

```
exports.goodbye = function() {
  console.log("bye!");
}
```

exports defines what require returns

app.js

```
var hello = require('./custom_hello');
var gb = require('./custom_goodbye');

hello();
gb.goodbye();
```

```
require('./custom_goodbye').goodbye();
```

If we only need to call once

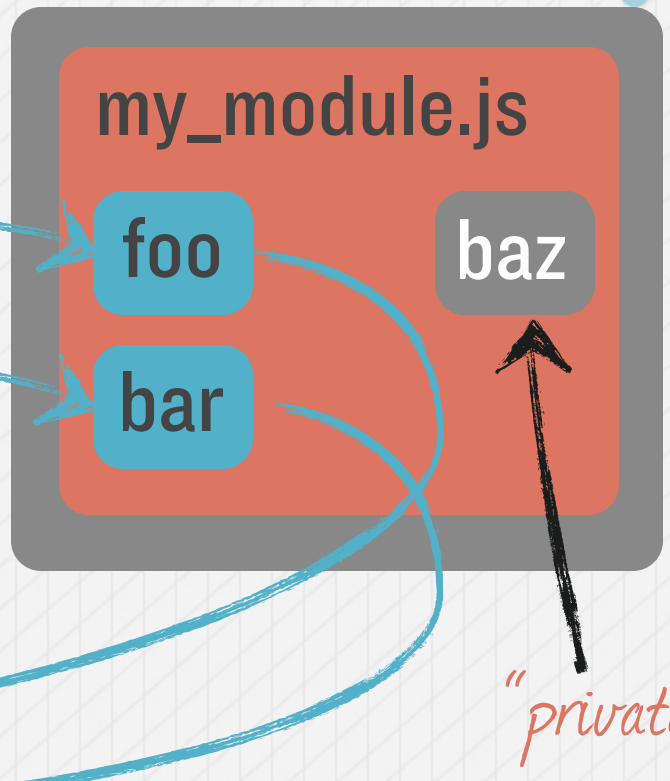
EXPORT MULTIPLE FUNCTIONS

my_module.js

```
var foo = function() { ... }
var bar = function() { ... }
var baz = function() { ... }

exports.foo = foo
exports.bar = bar
```

app.js

```
var myMod = require('./my_module');
myMod.foo();
myMod.bar();
```



MAKING HTTP REQUESTS

app.js

```
var http = require('http');

var message = "Here's looking at you, kid.";
var options = {
  host: 'localhost', port: 8080, path: '/', method: 'POST'
}

var request = http.request(options, function(response){
  response.on('data', function(data){
    console.log(data); logs response body
  });
});
request.write(message); begins request
request.end(); finishes request
```

ENCAPSULATING THE FUNCTION

app.js

```
var http = require('http');

var makeRequest = function(message) {
  var options = {
    host: 'localhost', port: 8080, path: '/', method: 'POST'
  }

  var request = http.request(options, function(response){
    response.on('data', function(data){
      console.log(data);
    });
  });
  request.write(message);
  request.end();
}

makeRequest("Here's looking at you, kid.");
```

CREATING & USING A MODULE

make_request.js

```
var http = require('http');

var makeRequest = function(message) {
   ...
}

exports = makeRequest;
```

app.js

```
var makeRequest = require('./make_request');

makeRequest("Here's looking at you, kid");
makeRequest("Hello, this is dog");
```

Where does require look for modules?

REQUIRE SEARCH

```
var make_request = require('./make_request')
var make_request = require('../make_request')
var make_request = require('/Users/eric/nodes/make_request')
```

look in same directory

look in parent directory

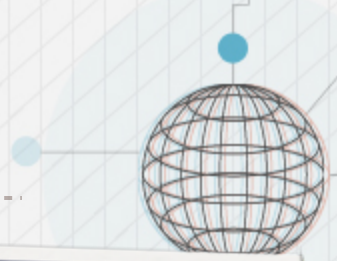
/Home/eric/my_app/app.js

Search in node_modules directories

```
var make_request = require('make_request')
```

- /Home/eric/my_app/node_modules/
- /Home/eric/node_modules/make_request.js
- /Home/node_modules/make_request.js
- /node_modules/make_request.js

NPM: THE USERLAND SEA



Package manager for node

- Comes with node
- Module Repository
- Dependency Management
- Easily publish modules
- “Local Only”

“Core” is small. “Userland” is large.

INSTALLING A NPM MODULE

In /Home/my_app

```
$ npm install request
```

---> https://github.com/mikeal/request

Installs into local node_modules directory

```
Home / my_app / node_modules / request
```

In /Home/my_app/app.js

```
var request = require('request');
```

Loads from local node_modules directory

LOCAL VS GLOBAL

Install modules with executables globally

```
$ npm install coffee-script -g
```

global

```
$ coffee app.coffee
```

Global npm modules can't be required

```
var coffee = require('coffee-script');
```

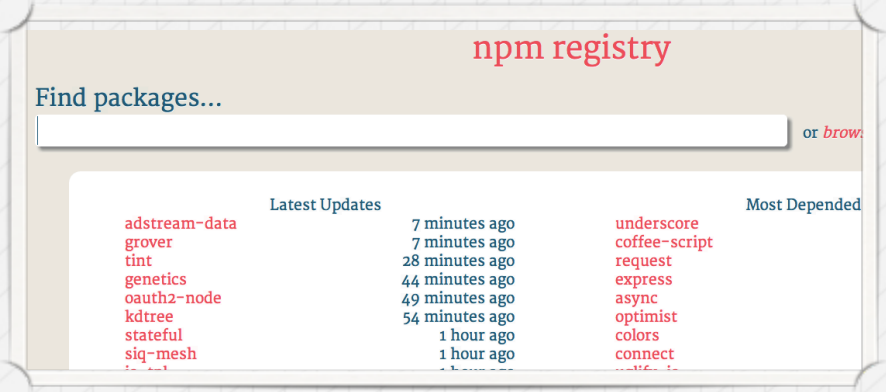
```
$ npm install coffee-script
```

Install them locally

```
var coffee = require('coffee-script');
```

FINDING MODULES

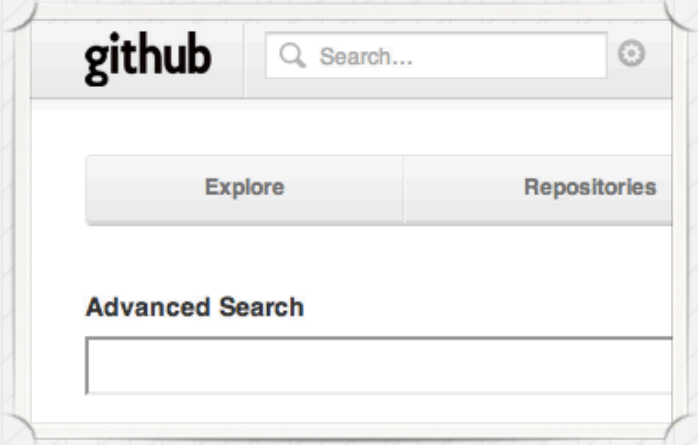
npm registry



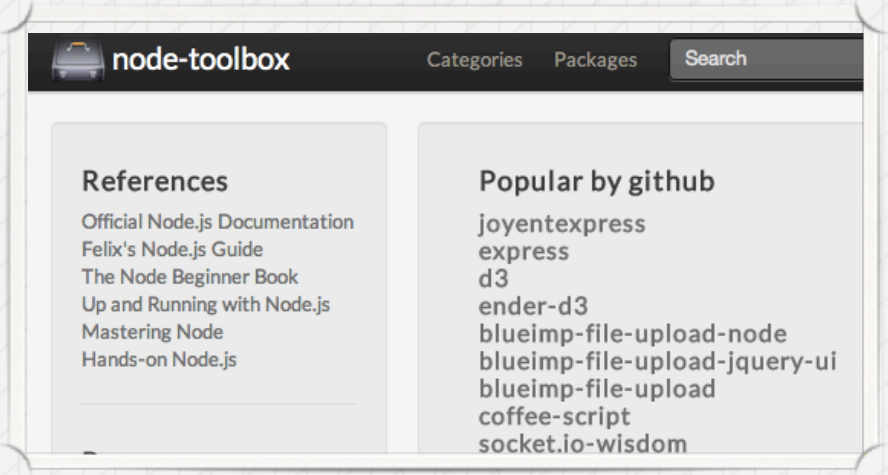
npm command line

```
$ npm search request
```

github search



toolbox.no.de



DEFINING YOUR DEPENDENCIES

my_app/package.json

```
{
  "name": "My App",
  "version": "1",
  "dependencies": {
    "connect": "1.8.7"
  }
}
```

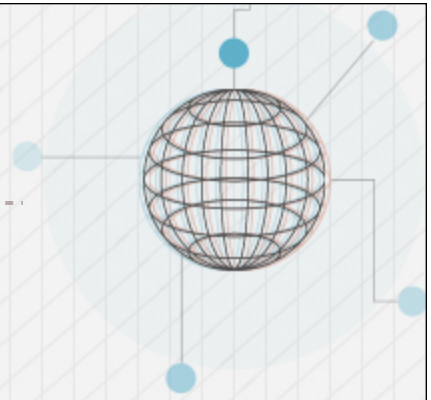
version number

```
$ npm install
```

Installs into the node_modules directory

my_app / node_modules / connect

DEPENDENCIES



my_app/package.json

```
"dependencies": {
  "connect": "1.8.7"
}
```

No conflicting modules!

Installs sub-dependencies

my_app / node_modules / connect

connect / node_modules / qs

connect / node_modules / mime

connect / node_modules / formidable

SEMANTIC VERSIONING

Major Minor Patch

```
"connect": "1.8.7"
```

1 . 8 . 7

Ranges

`"connect": "~1"`	--> `>=1.0.0 <2.0.0`	Dangerous
`"connect": "~1.8"`	--> `>=1.8 <2.0.0`	API could change
`"connect": "~1.8.7"`	--> `>=1.8.7 <1.9.0`	Considered safe

http://semver.org/

EXPRESS

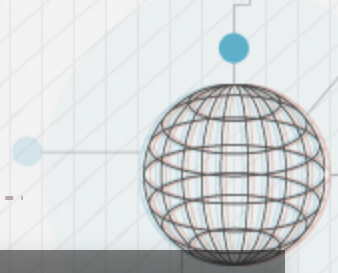
- LEVEL FIVE -

EXPRESS

"Sinatra inspired web development framework for Node.js -- insanely fast, flexible, and simple"

- Easy route URLs to callbacks
- Middleware (from Connect)
- Environment based configuration
- Redirection helpers
- File Uploads

INTRODUCING EXPRESS



```
$ npm install express
```

```
var express = require('express');
```

```
var app = express.createServer();
```

root route

```
app.get('/', function(request, response) {
  response.sendfile(__dirname + "/index.html");
});
```

current directory

```
app.listen(8080);
```

```
$ curl http://localhost:8080/
> 200 OK
```

EXPRESS ROUTES

app.js

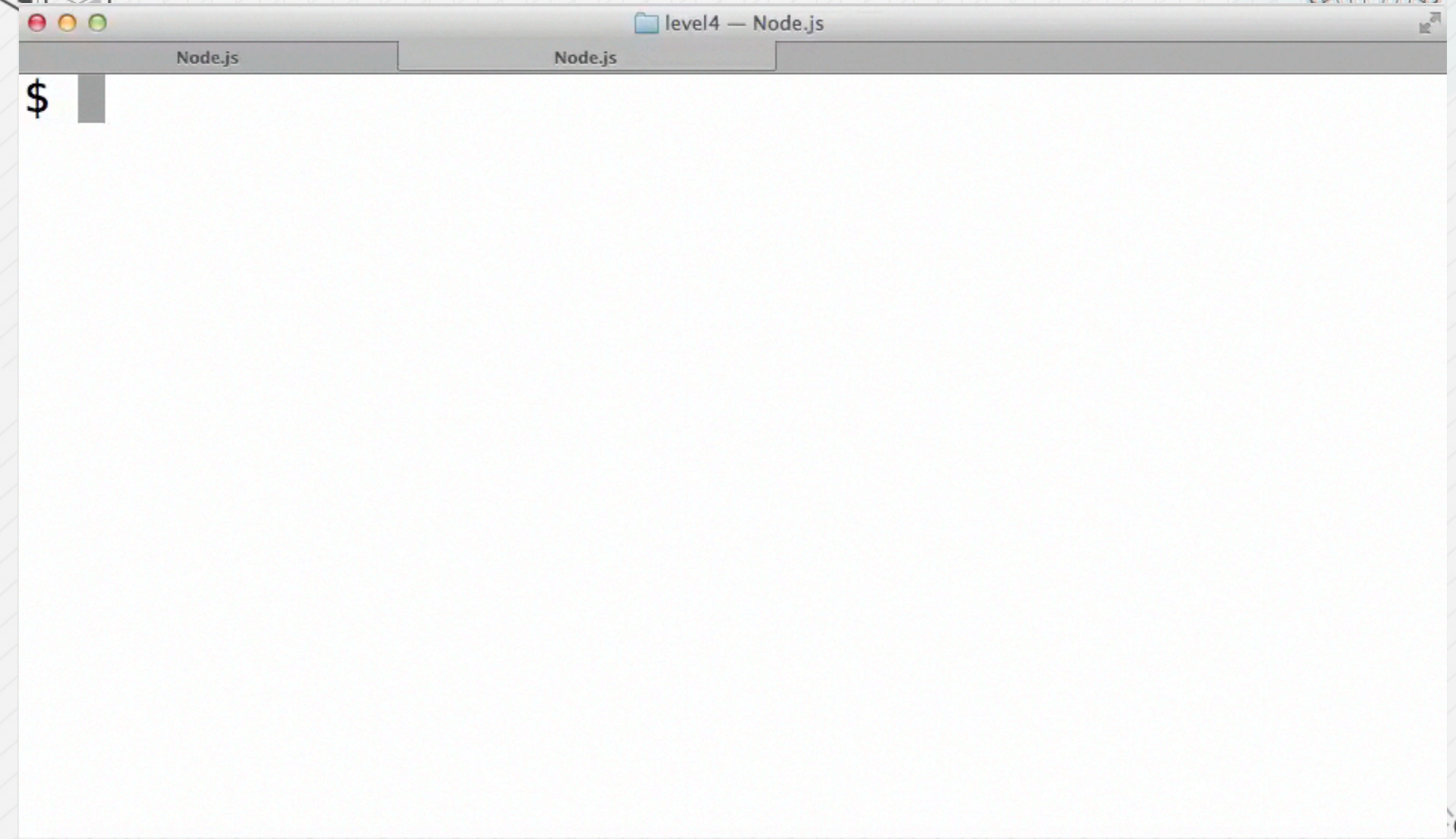
```
var request = require('request');
var url = require('url');

app.get('/tweets/:username', function(req, response) {

  var username = req.params.username;

  options = {
    protocol: "http:",
    host: 'api.twitter.com',
    pathname: '/1/statuses/user_timeline.json',
    query: { screen_name: username, count: 10}
  }

  var twitterUrl = url.format(options);
  request(twitterUrl).pipe(response);
});
```

route definition

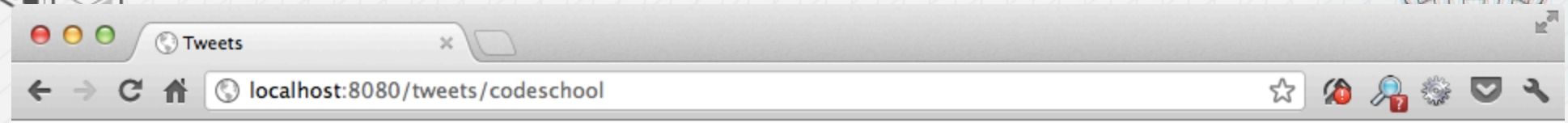
get the last 10 tweets for screen_name

pipe the request to response

EXPRESS ROUTES



EXPRESS + HTML



Tweets for @codeschool

- @seandevineinc let us know how it goes, and good luck
- @larzconwell Nope, we didn't give those away. The one David has comes from http://t.co/XrvybxnS ^OL
- We just released a new Code TV screencast for enrolled members. Part of 1 of @markkendall's jQuery Mobile series. http://t.co/FstmuYEM
- ^vc
- We also have stickers..
- Are you at Railsconf? Come by the beginner track room.. we're giving away free Rails for Zombies T-shirts (while they last) #railsconf
- We themed out our Code School store. Check it out http://t.co/VOZCgorM ^vc
- Have you gotten your Code School & Zombies t-shirts yet? Check out our $19 sale this week... http://t.co/b7JUMfxy ^vc

EXPRESS TEMPLATES

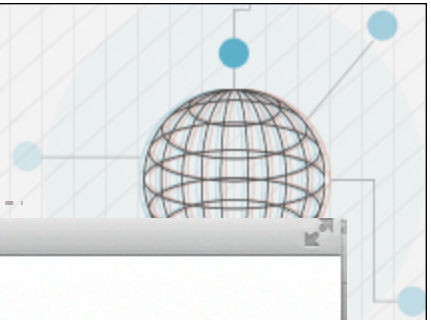
app.js

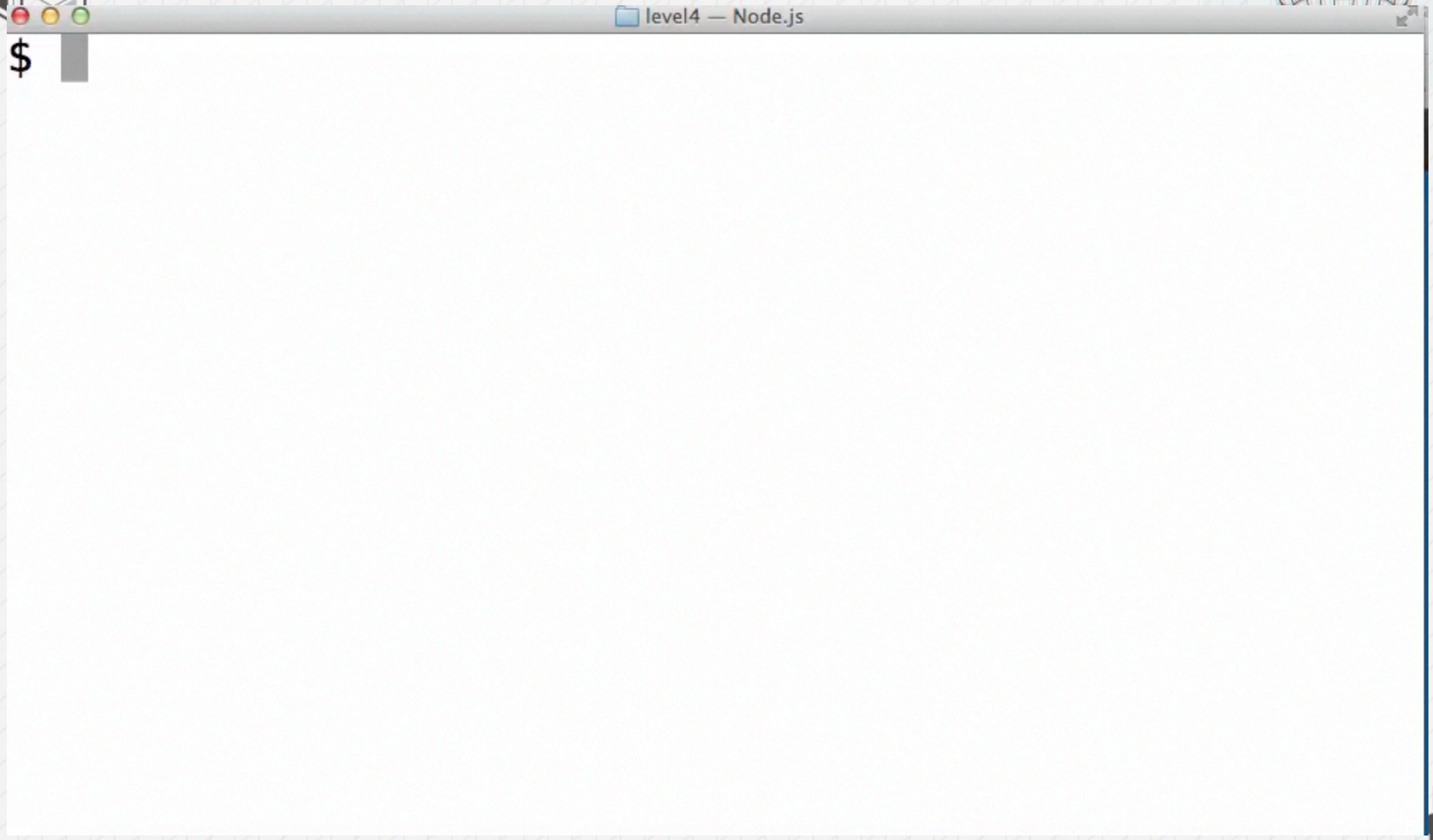
```
app.get('/tweets/:username', function(req, response) {
  ...
  request(url, function(err, res, body) {
    var tweets = JSON.parse(body);
    response.render('tweets.ejs', {tweets: tweets, name: username});
  });
});
```

tweets.ejs

```
<h1>Tweets for @<%= name %></h1>
<ul>
  <% tweets.forEach(function(tweet){ %>
    <li><%= tweet.text %></li>
  <% }); %>
</ul>
```

EXPRESS TEMPLATES





TEMPLATE LAYOUTS

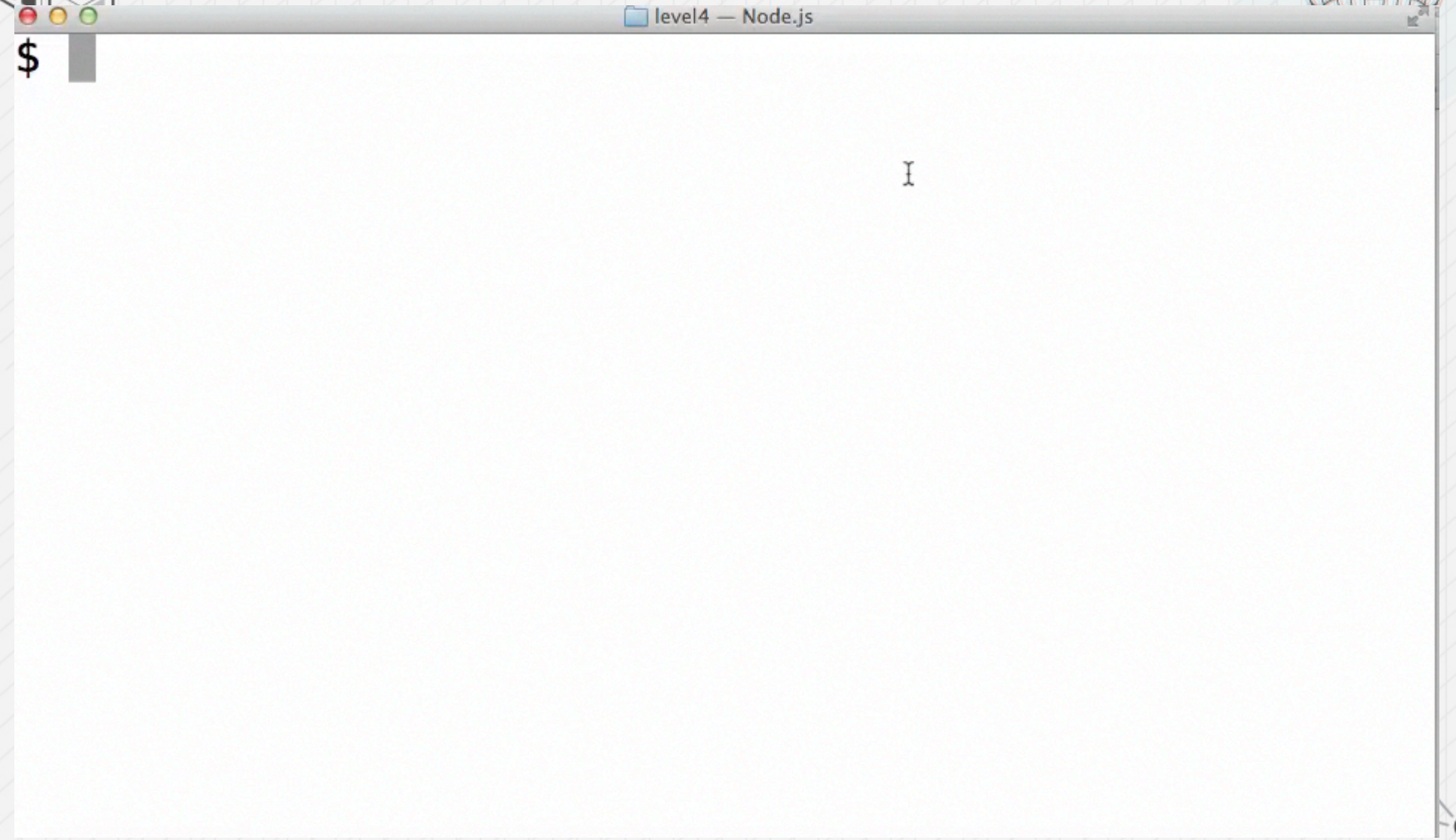
tweets.ejs

```
<h1>Tweets for @<%= name %></h1>
<ul>
  <% tweets.forEach(function(tweet){ %>
    <li><%= tweet.text %></li>
  <% }); %>
</ul>
```

layout.ejs

```
<!DOCTYPE html>
<html>
  <head>
    <title>Tweets</title>
  </head>
  <body>
    <%- body %>
  </body>
</html>
```

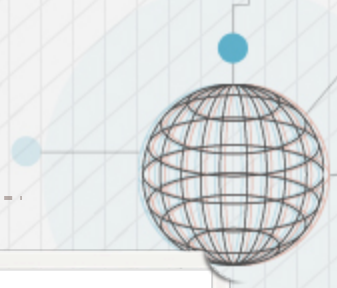
EXPRESS TEMPLATES



SOCKET.IO

- LEVEL SIX -

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?

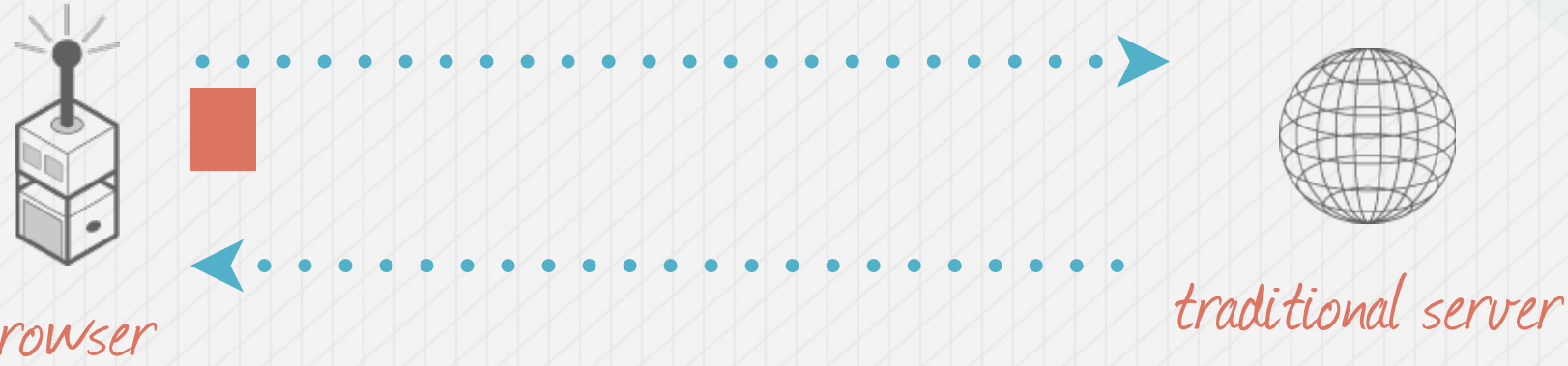
Derrick *joined the room*

Type your message

SEND

SOCKET.IO

WEBSOCKETS



Traditional request/response cycle

WEBSOCKETS

browser

socket.io

Using duplexed websocket connection

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

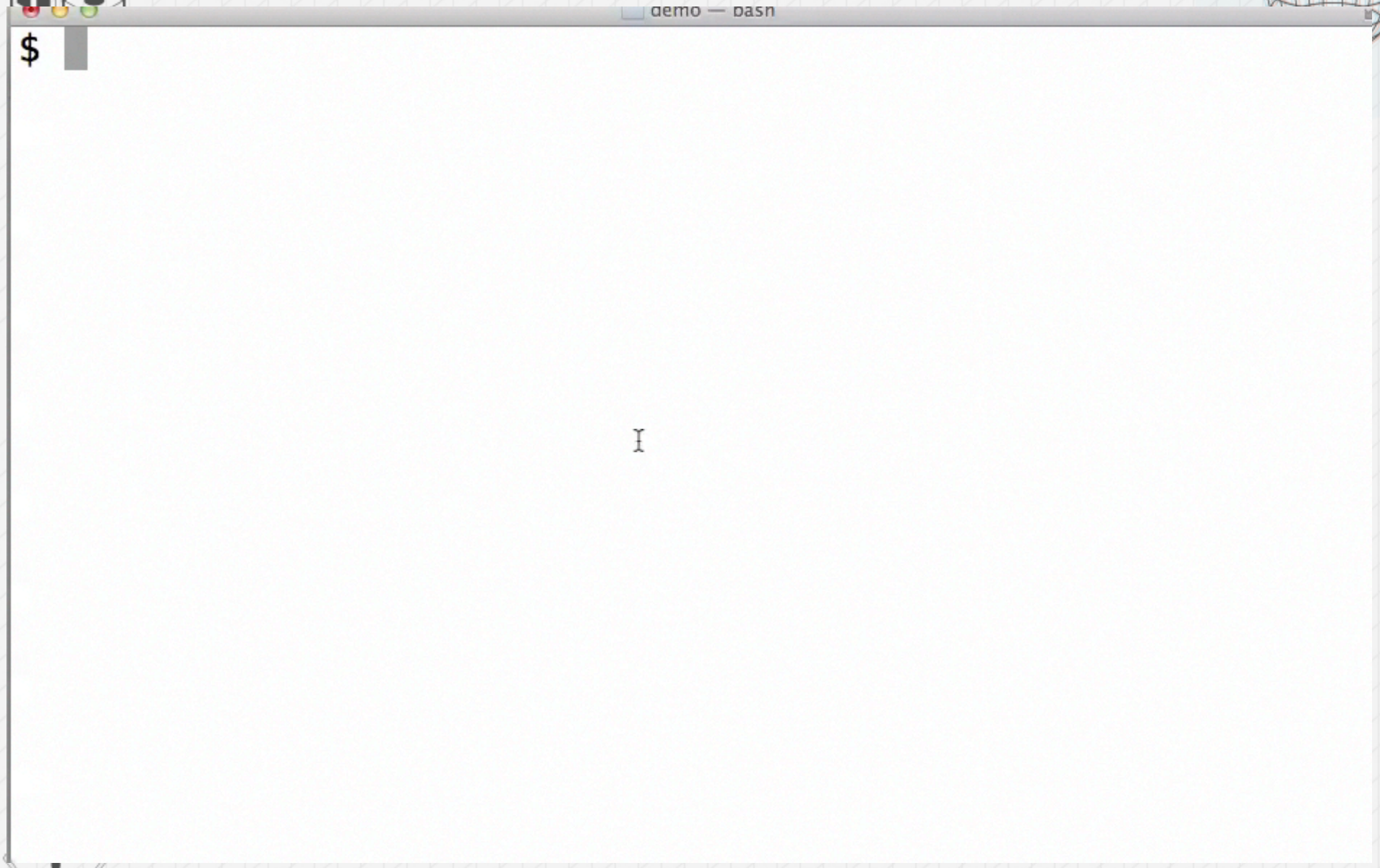
```
io.sockets.on('connection', function(client) {
  console.log('Client connected...');

  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

CHATTR HELLO WORLD



SENDING MESSAGES TO SERVER

app.js

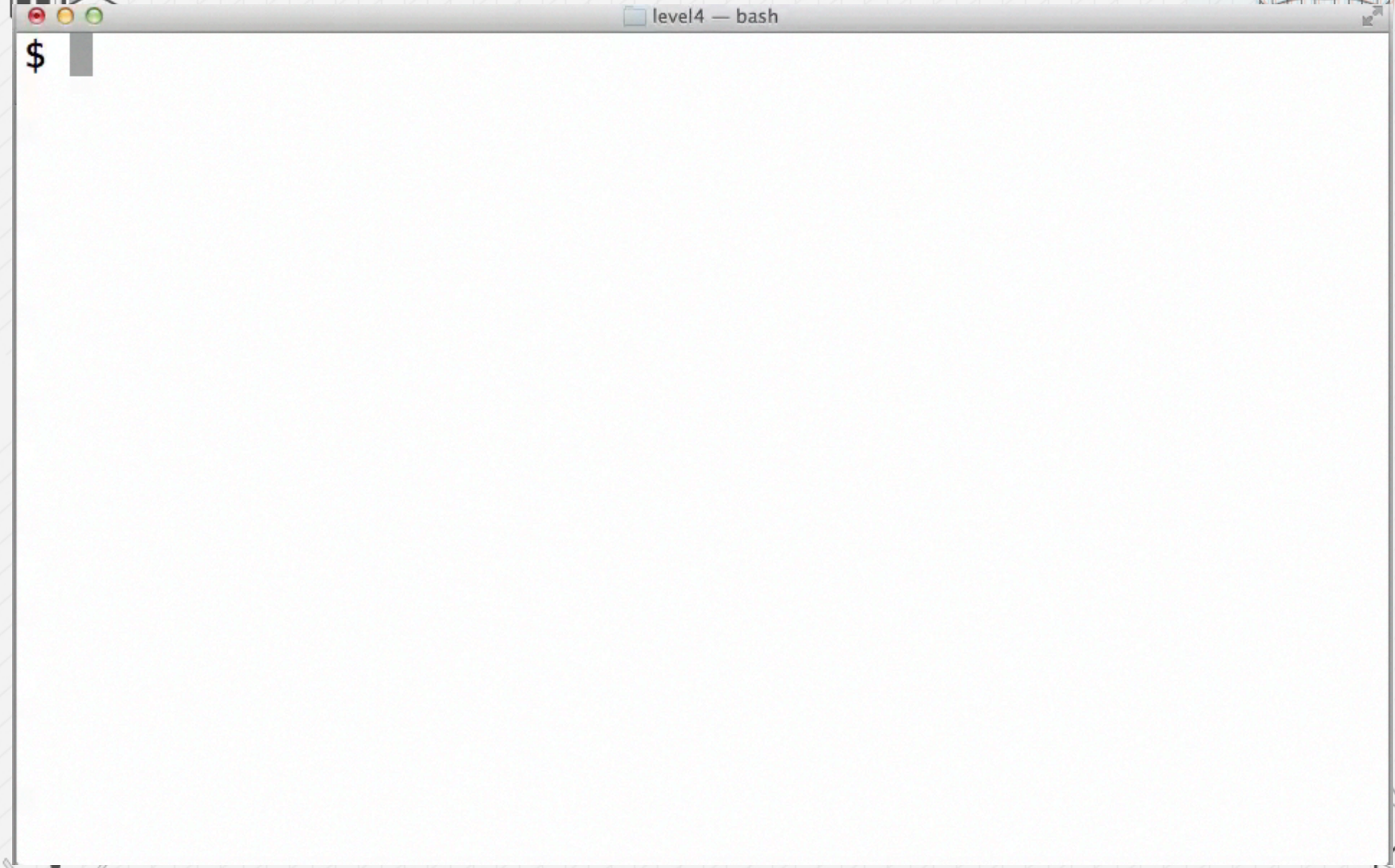
```
io.sockets.on('connection', function(client) {
  client.on('messages', function (data) {
    console.log(data);
  });
});
```

listen for 'messages' events

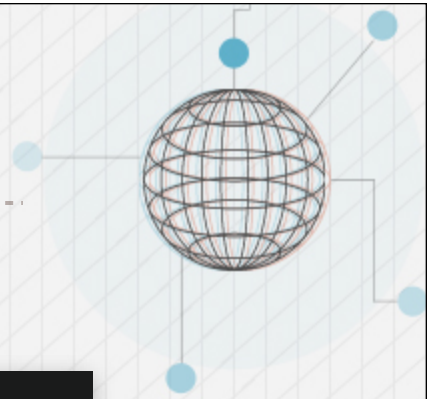
index.html

```
<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>
```

CHATTR HELLO WORLD

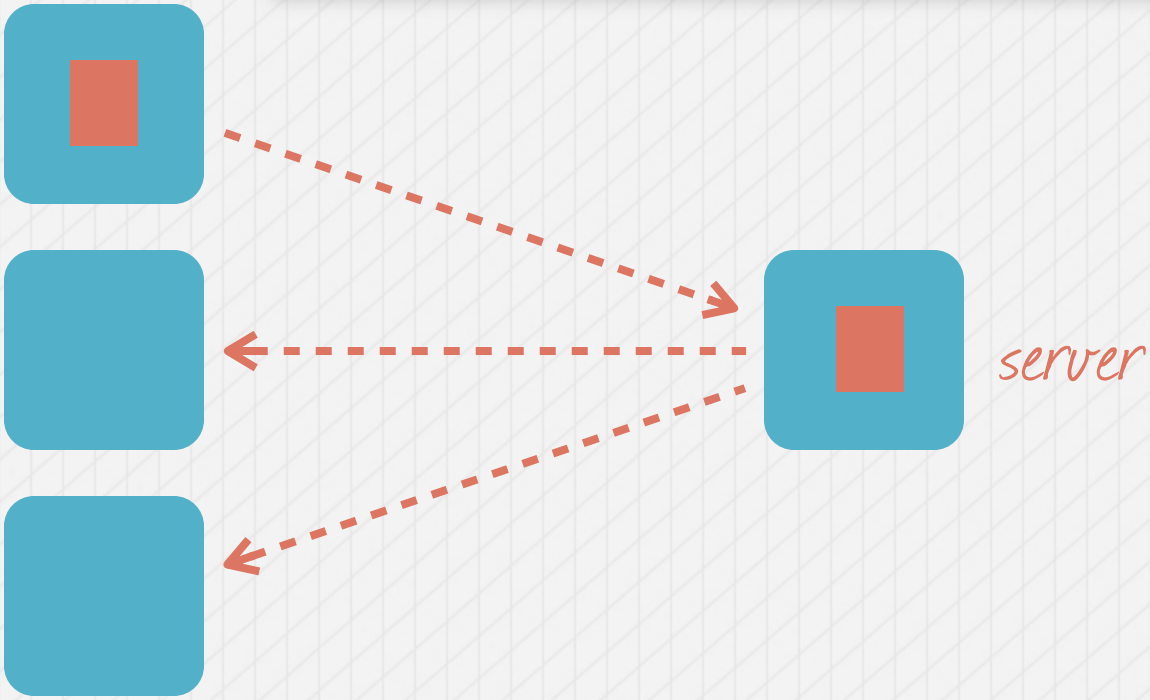


BROADCASTING MESSAGES



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);
  });
});
```

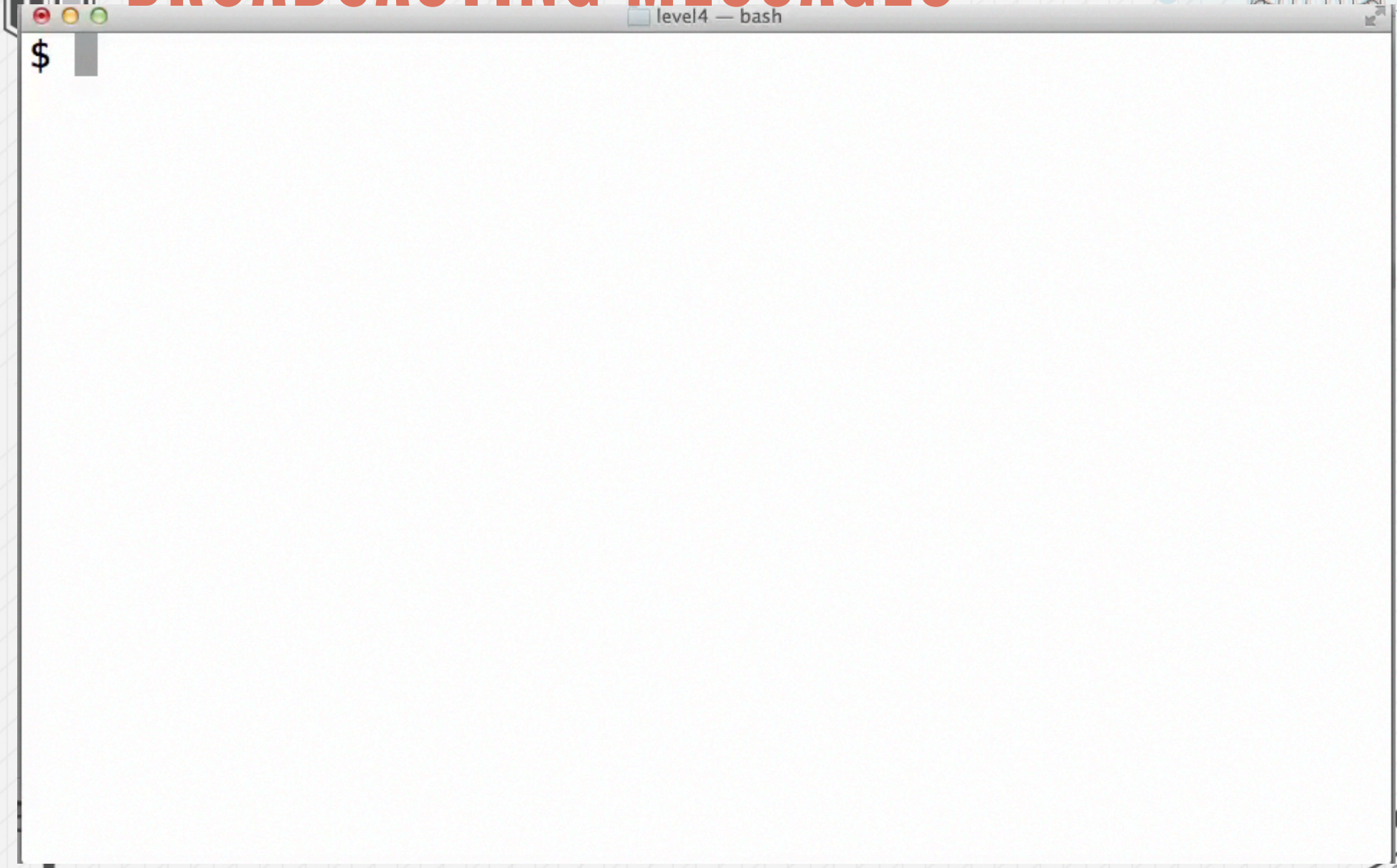
broadcast message to all other clients connected

index.html

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

insert message into the chat

BROADCASTING MESSAGES



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

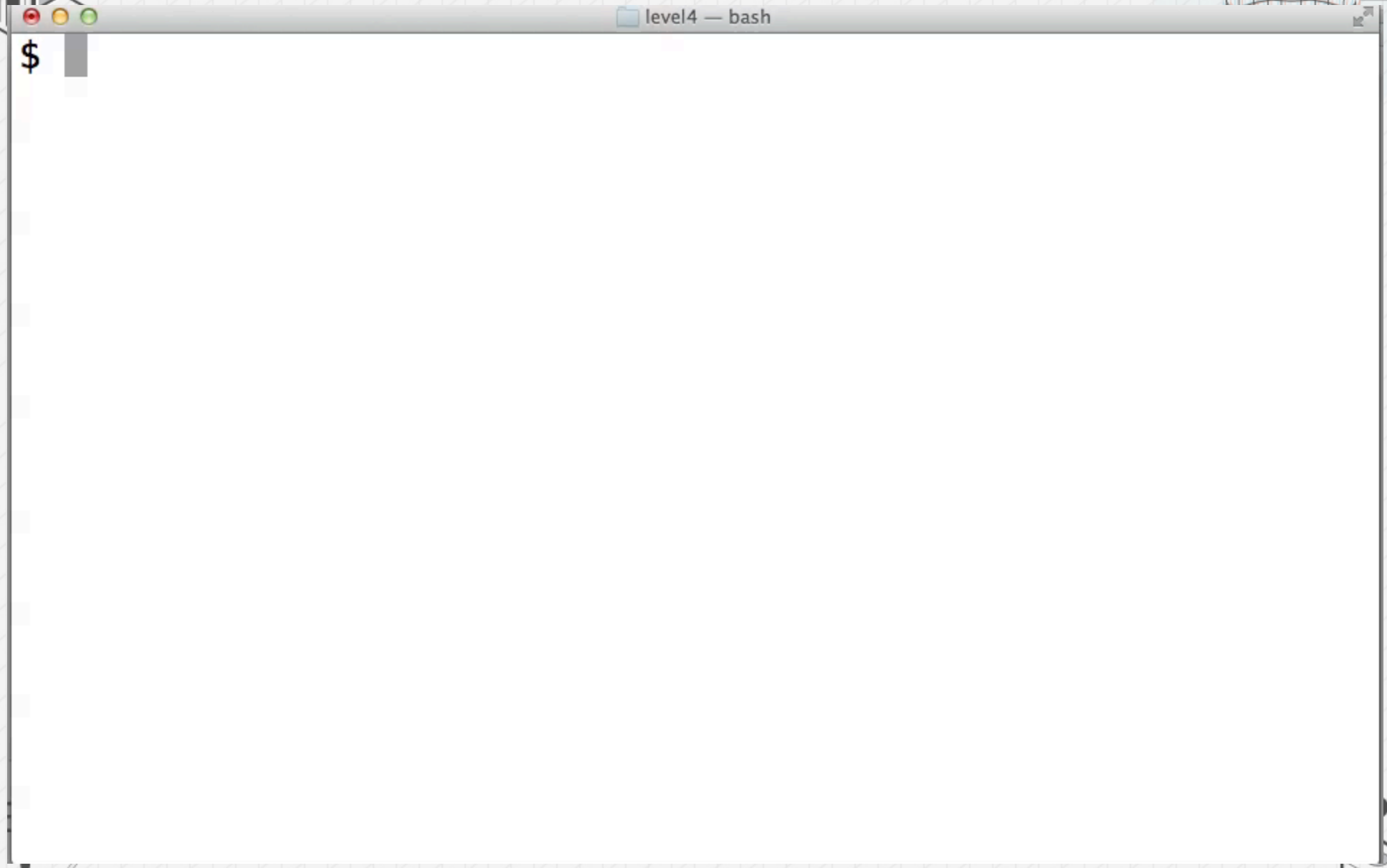
app.js

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

set the nickname associated with this client

broadcast with the name and message

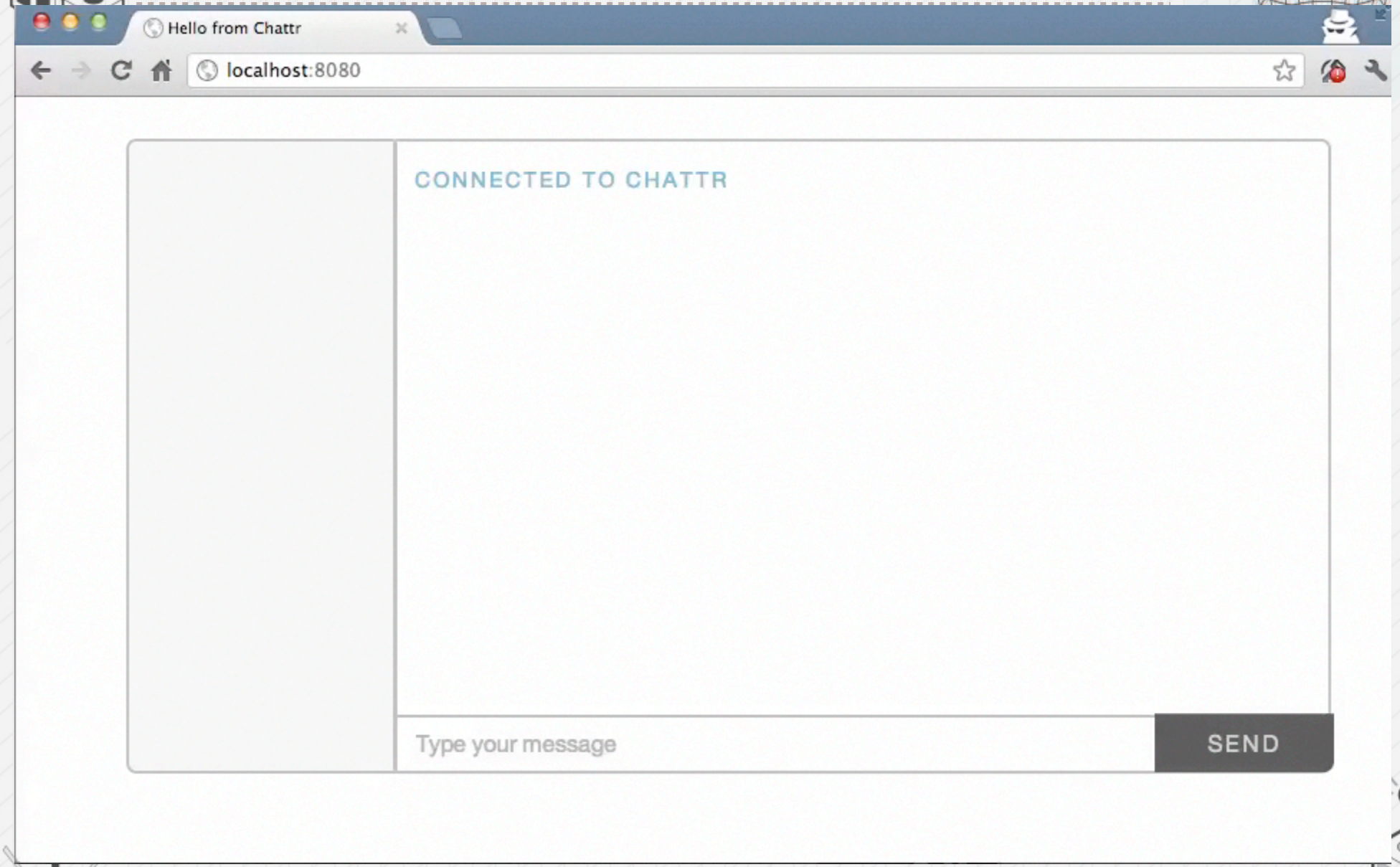
SAVING DATA ON THE CLIENT



PERSISTING DATA

- LEVEL SEVEN -

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
});
```

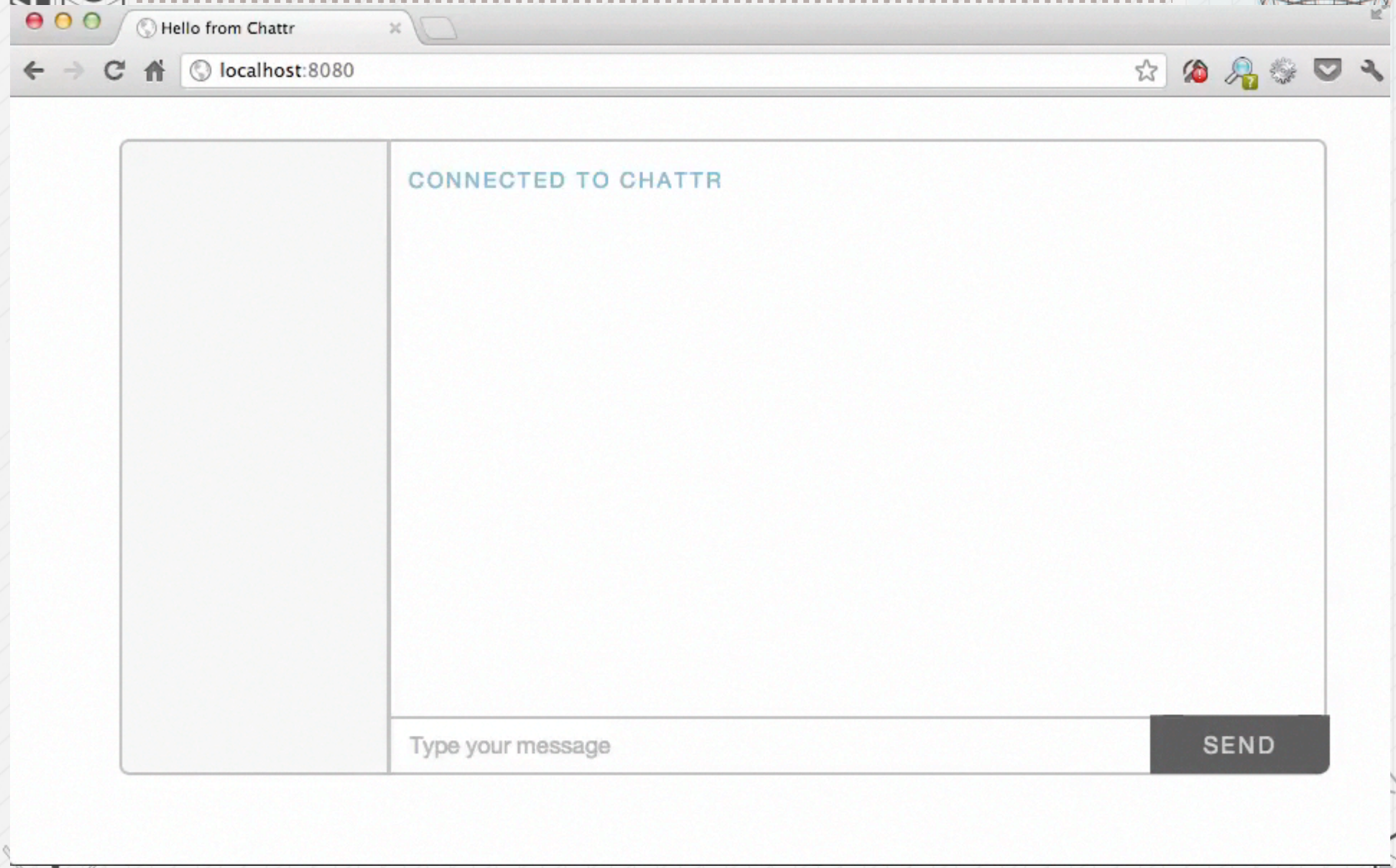
EMITTING MESSAGES

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

RECENT MESSAGES



PERSISTING STORES

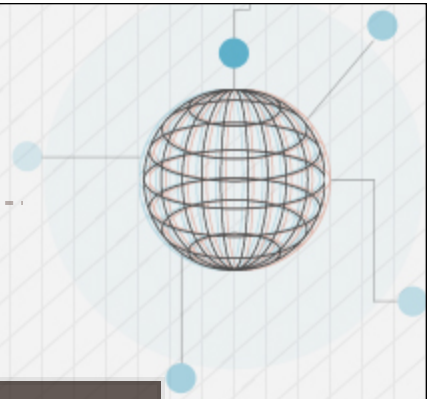
- MongoDB
- CouchDB
- PostgreSQL
- Memcached
- Riak

All non-blocking!

redis

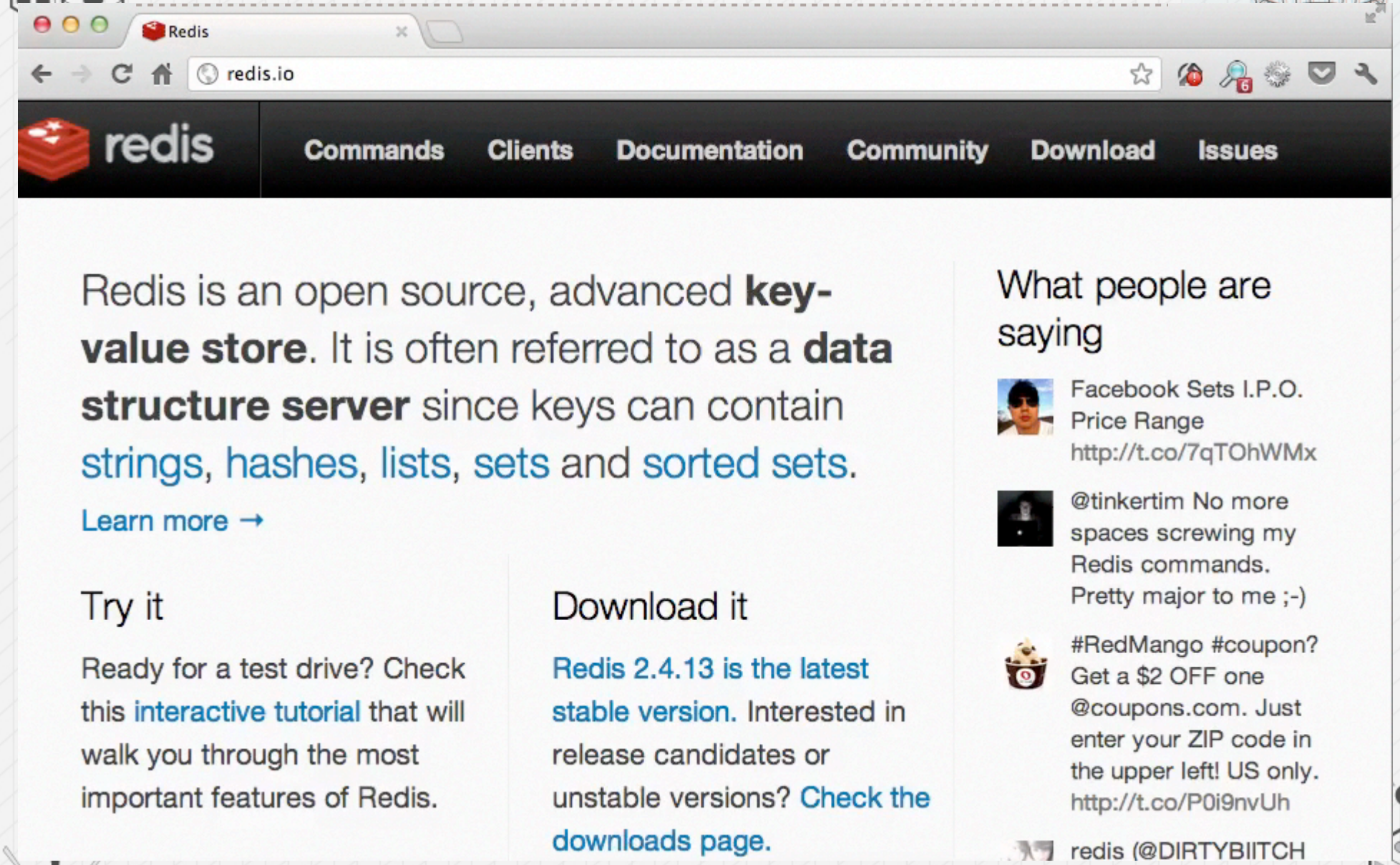
Redis is a key-value store

REDIS DATA STRUCTURES

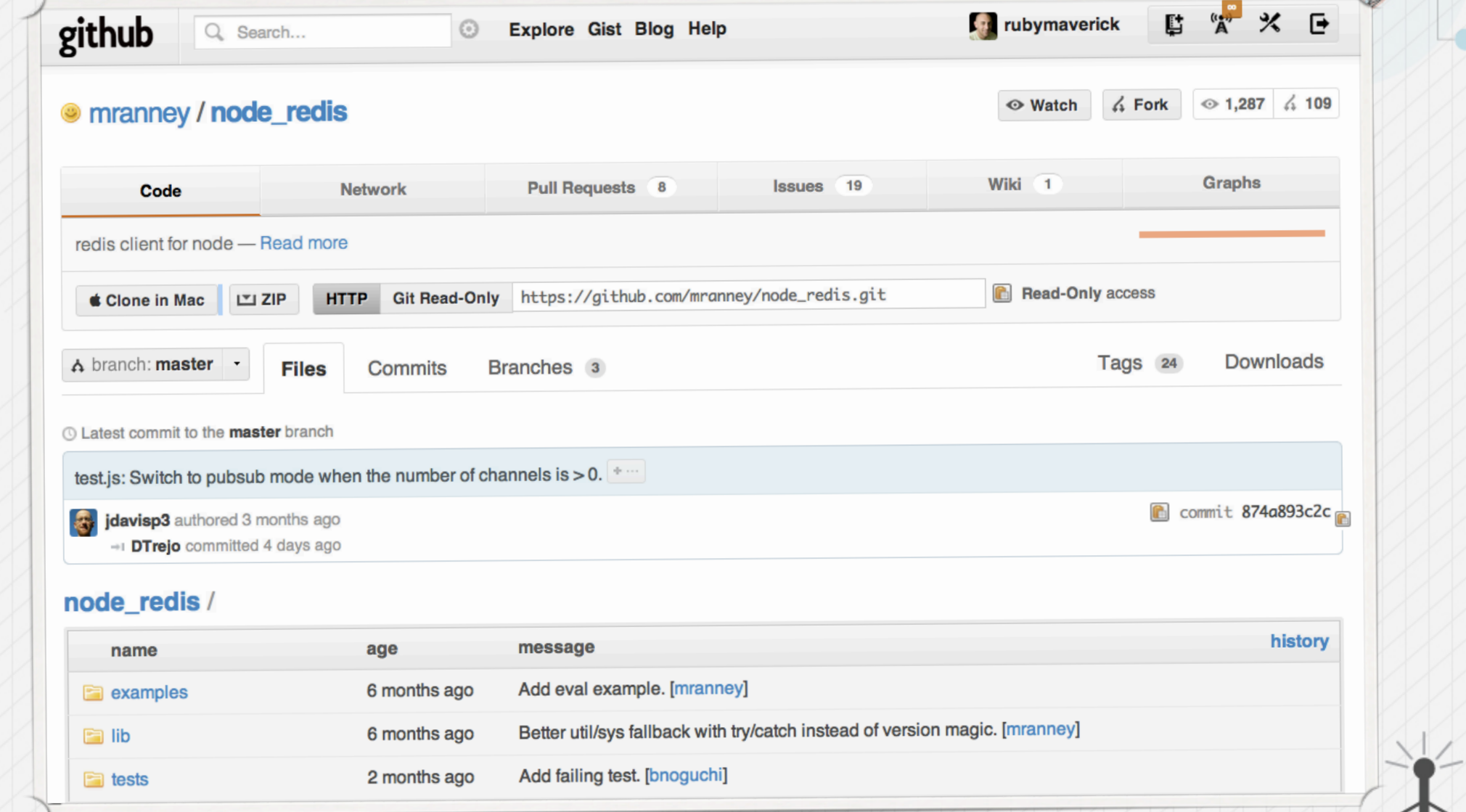


data structure	commands
Strings	SET, GET, APPEND, DECR, INCR...
Hashes	HSET, HGET, HDEL, HGETALL...
Lists	LPUSH, LREM, LTRIM, RPOP, LINSERT...
Sets	SADD, SREM, SMOVE, SMEMBERS...
Sorted Sets	ZADD, ZREM, ZSCORE, ZRANK...

REDIS COMMAND DOCUMENTATION

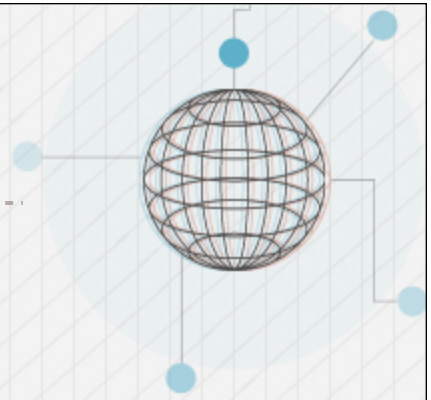


NODE REDIS



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

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});
                  need to turn object into string to store in redis

  redisClient.lpush("messages", message, function(err, response) {
    redisClient.ltrim("messages", 0, 10);
  });
}
```

keeps newest 10 items

OUTPUT FROM LIST

app.js

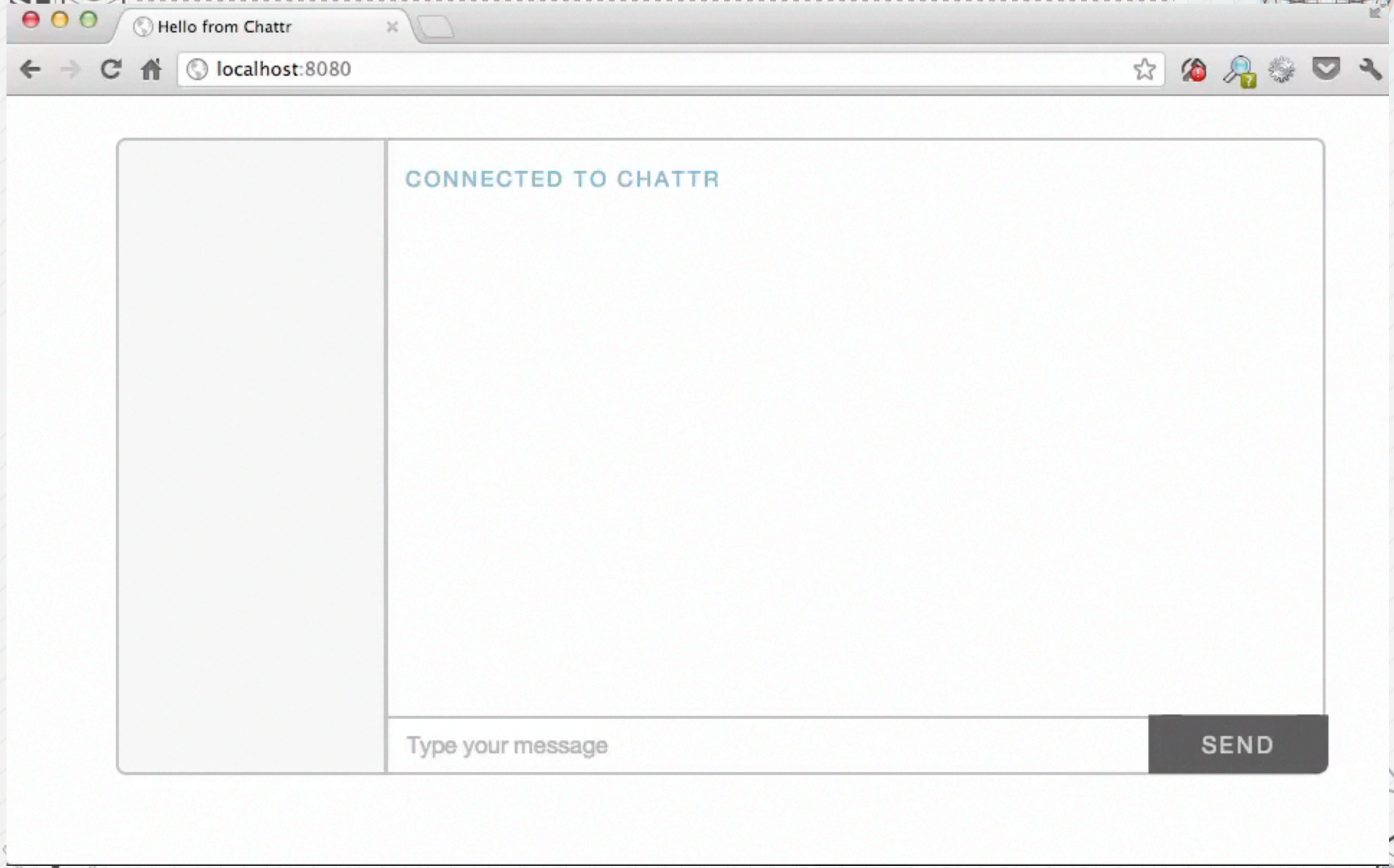
```
client.on('join', function(name) {
  messages.forEach(function(message) {
    client.emit("messages", message.name + ": " + message.data);
  });
});
```

OUTPUT FROM LIST

app.js

```
client.on('join', function(name) {
  redisClient.lrange("messages", 0, -1, function(err, messages){
    messages = messages.reverse();    reverse so they are emitted
                                        in correct order

    messages.forEach(function(message) {
      message = JSON.parse(message);  parse into JSON object
      client.emit("messages", message.name + ": " + message.data);
    });
  });
});
```

IN ACTION


Hello from Chattr
localhost:8080
CONNECTED TO CHATTR
Type your message
SEND

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();
}
```