44-563: Unit 06

Developing Web Applications and Services

Includes

- Node
- Express
- Socket.io
- Chat App
- Friday quiz

Node.js platform

- Software **platform** for scalable, event-driven, i/o-focused server-side and networking applications
- https://nodejs.org/
- API: https://nodejs.org/api/
- To execute a module in a file named app.js, run:

```
> node app.js
```

Modules in Node

In Node.js:

- Each file is its own **module**
- To make functions / objects in one module available for use in others, assign them to the former's **module.exports** variable
- To import a module, use **require()**
- By default, module.exports = { }, i.e., an empty **object**.
- Assigning a value attaches a new property to that object, an idea studied previously.

```
// in foo.js
module.exports.PI = 3.141592654
module.exports.sumSquares = function(a,b){return a*a + b*b}
```

```
// in foo_user.js:
let foo = require('./foo')

let whatIsADeliciousDessertTreat = foo.PI
let findSumSquares = foo.sumSquares(3,4)
```

https://team.goodeggs.com/export-this-interface-design-patterns-for-node-js-modules-b48a3b1f8f40#.8heed0gz0

Modules in Node

If you assign a **function** to module.exports in its entirety, e.g.,

- module.exports = **function** area(width,height){return width*height} // in foo.js

then you can use it as follows:

```
let foo = require('./foo')
let area2 = foo(3,4) // equivalent since there is only one function
console.log("The area is " + area2)
```

Rather than export individual functions and constants, you can also **export an entire object**. This is the preferred way of doing things.

```
// in foo.js
let geometer = {pi:3.14159, area:function(width,height){return width*height}}
module.exports = geometer
```

This is an example of a singleton. What's a singleton? Oh, about 2,000 pounds □□

```
let foo = require('./foo')
console.log(foo.pi)
console.log(foo.area(5,6))
```

https://bitbucket.org/professorcase/weathernode

Clone the above repo ...

weatherReader.js

```
let http = require('http')

function printWeather(city, weather) {
    console.log('In ' + city + ', it is ' + weather + ' degrees C.')
}
function printError(error) {  console.error(error.message) }

module.exports = function get(city){  // makes it public
    let request = http.get('http://api.openweathermap.org/data/2.5/weather?q='+
    city + '&units=metric&apikey=c184205bc1fcbcdc42c4b37ccf710de3', responseFunction)

    function responseFunction(response) {
        let body = ''
        response.on('data', function(chunk) { body += chunk }) // on getting data
        response.on('end', function() {  // on completion, do this
            if (response.statusCode === 200) {
                try {
                    var weatherAPI = JSON.parse(body)
                    printWeather(weatherAPI.name, weatherAPI.main.temp)
                } catch(error) {
            } else {
                printError({messa

            }
        })
  }
  request.on('error', printError)
}
```

To make a **single function** available to other files, assign the function to the **module.exports** object.

Now, we can **import** from this file with

```
var noaa =  require('./weatherReader.js')
```

Since noaa is now the function, we must invoke it by passing our argument:

```
Var w = noaa('London')
```

getWeather.js (our app that calls our library of code)

```
// invoke single function
var noah = require('./weatherReader.js')
noah('Maryville, Missouri')

// invoke single function directly
require('./weatherReader.js')('Maryville, Missouri')
```

Git Bash here (or Open Cmd Window Here as Administrator):

```
$   node getWeather.js
```

Or

```
$ node getWeather
```

weatherReader2.js

```
var api = {
    getTemp: function(city) {   },
    getHumidity: function(city) { }
}
module.exports = api
```

To make **multiple** functions available to other files, group them in an object, and assign the object to **module.exports**.

We still **import** this file with

```
var noaa =  require('./weatherReader2.js')
```

Since what we exported is now an object, we invoke functions by their key and by passing their arguments:

```
var noaa = require('./weatherReader2.js')
noaa.getTemp('Maryville, MO')
noaa.getHumidity('Maryville, MO')
```

weatherReader.js

```
let http = require('http')

function printWeather(city, weather) {
    console.log('In ' + city + ', it is ' + weather + ' degrees C.')
}
function printError(error) {  console.error(error.message) }

module.exports = function get(city){  // makes it public
    let request = http.get('http://api.openweathermap.org/data/2.5/weather?q='+
    city + '&units=metric&apikey=c184205bc1fcbcdc42c4b37ccf710de3', responseFunction)

    function responseFunction(response) {
        let body = ''
        response.on('data', function(chunk) { body += chunk }) //On getting data
        response.on('end', function() {  // on completion, do this
            if (response.statusCode === 200) {
                try {
                    var weatherAPI = JSON.parse(body) // JSON is built-in to JS
                    printWeather(weatherAPI.name, weatherAPI.main.temp)
                } catch(error) {  printError(error)   }
            } else {
                printError({message: 'Error getting weather from ' + city + '. (' +
                                     http.STATUS_CODES[response.statusCode] + ')'})
            }
        })
  }
  request.on('error', printError)  // on getting an error, do this
}
```

To build a basic node app, first require the http module.

weatherReader.js

```
let http = require('http')

function printWeather(city, weather) {
    console.log('In ' + city + ', it is ' + weather + ' degrees C.')
}
function printError(error) {  console.error(error.message) }

module.exports = function get(city){
    let request = http.get('http://api.openweathermap.org/data/2.5/weather?q='+
    city + '&units=metric&apikey=c184205bc1fcbcdc42c4b37ccf710de3', responseFunction)

    function responseFunction(response) {
        let body = ''
        response.on('data', function(chunk) { body += chunk }) //On getting data
        response.on('end', function() {  // on completion, do this
            if (response.statusCode === 200) {
                try {
                    var weatherAPI = JSON.parse(body)
                    printWeather(weatherAPI.name, weatherAPI.main.temp)
                } catch(error) {  printError(error)   }
            } else {
                printError({message: 'Error getting weather from ' + city + '. (' +
                                     http.STATUS_CODES[response.statusCode] + ')'})
            }
        })
  }
  request.on('error', printError)  // on getting an error, do this
}
```

1. When we call http.get(), what is the callback?
2. When will the callback be executed?

weatherReader.js

```
let http = require('http')

function printWeather(city, weather) {
    console.log('In ' + city + ', it is ' + weather + ' degrees C.')
}
function printError(error) {  console.error(error.message) }

module.exports = function get(city){
    let request = http.get('http://api.openweathermap.org/data/2.5/weather?q='+
    city + '&units=metric&apikey=c184205bc1fcbcdc42c4b37ccf710de3', responseFunction)

    function responseFunction(response) {
        let body = ''
        response.on('data', function(chunk) { body += chunk }) //On getting data, do this
        response.on('end', function() {  // on completion, do this
            if (response.statusCode === 200) {
                try {
                    var weatherAPI = JSON.parse(body)
                    printWeather(weatherAPI.name, weatherAPI.main.temp)
                } catch(error) {  printError(error)   }
            } else {
                printError({messa

            }
        })
  }
  request.on('error', printError)
}
```

1. Node's EventEmitters have an **on()** method that binds an eventHandler to an event string name (much like jQuery).
2. What function is bound to the 'data' event?
3. What function is bound to the 'end' event?
4. What function is bound to the 'error' event?
5. How many nested functions appear here?

A Question of Timing

http.get(*path*,*callback*) is **asynchronous**: by the time it finishes the data has *not* arrived. All that http.get() does is a) set up a response stream, b) establish the (nested) callback function, passing in the response (as well as request), and then end.

The callback function continues to live on, gathering in data as it arrives and processing it.

http://stackoverflow.com/questions/26874565/nodejs-listening-to-response-stream-on-data-event-from-http-get

Http API used

http.request(options, callback)

http.**get**(options, callback) // most common request

Request events

- **error** - emitted when an error is detected in the request

Response events

- **data** - Emitted when a piece of the message body is received.
- **end** - Emitted exactly once for each request. After that, no more `'data'` events will be emitted on the request.

In addition to listening for events - we can **raise** or **emit** our own **custom** events as needed.

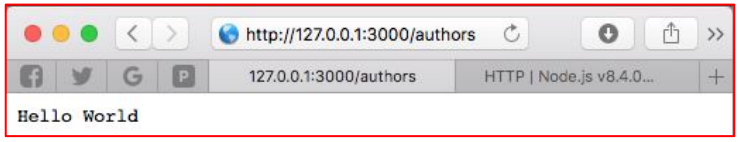
https://nodejs.org/api/http.html

https://millermedeiros.github.io/mdoc/examples/node_api/doc/http.html

A Web Server in Node.js

Creating a rudimentary web server in Node takes 2 easy-as-  steps:

1. Invoke http.createServer() with an appropriate callback
2. Start listening for responses




```
Server running at http://127.0.0.1:3000/
Method: GET - URL: /authors
```

1. How would you modify this to return **different** content (not just 'Hello World\n') for different URLS, e.g.,
   a. /publishers
   b. /booksellers
   c. /authors/fiction
   d. Etc. ?
2. Do you think there should be a better way?

```
const http = require('http')

const hostname = '127.0.0.1'
const port = 3000

const server = http.createServer((req, res) => {
  console.log("Method: " + req.method + " - URL: " +  req.url)
  res.statusCode = 200
  res.setHeader('Content-Type', 'text/plain')
  res.end('Hello World\n')
})

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`)
})
```

Express framework

- Fast, unopinionated, minimalist web **framework** for Node.js
- Used to create **web apps** on the Node.js platform, provides helpful objects
- https://expressjs.com/
- API: static(), Router(), Application, Request, Response, Router objects
- API: http://expressjs.com/en/api.html

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

Express framework

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

- app represents the Express application.
- It is a JavaScript **object** that will be passed to Node.js HTTP servers (cf. the source, line 616) as a callback to handle requests (app is our requestListener callback)
- It contains methods to
  - route requests -- app.get(), app.post()
  - add middleware -- app.use()
  - render html views -- app.render()

Check out Evan Hahn's delightfully clear and detailed explanation

Express hello world app

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

app.get('/', function (req, res) {
  res.send('Hello World!')
})

app.listen(3000, function () {
  console.log('Example app listening on port
3000!')
})
```

https://expressjs.com/en/starter/hello-world.html

Explore

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

app.get('/', function (req, res) {
  res.send('Hello World!')
})

app.listen(3000, function () {
  console.log('Example app listening on
port 3000!')
})
```

https://expressjs.com/en/starter/hello-world.html

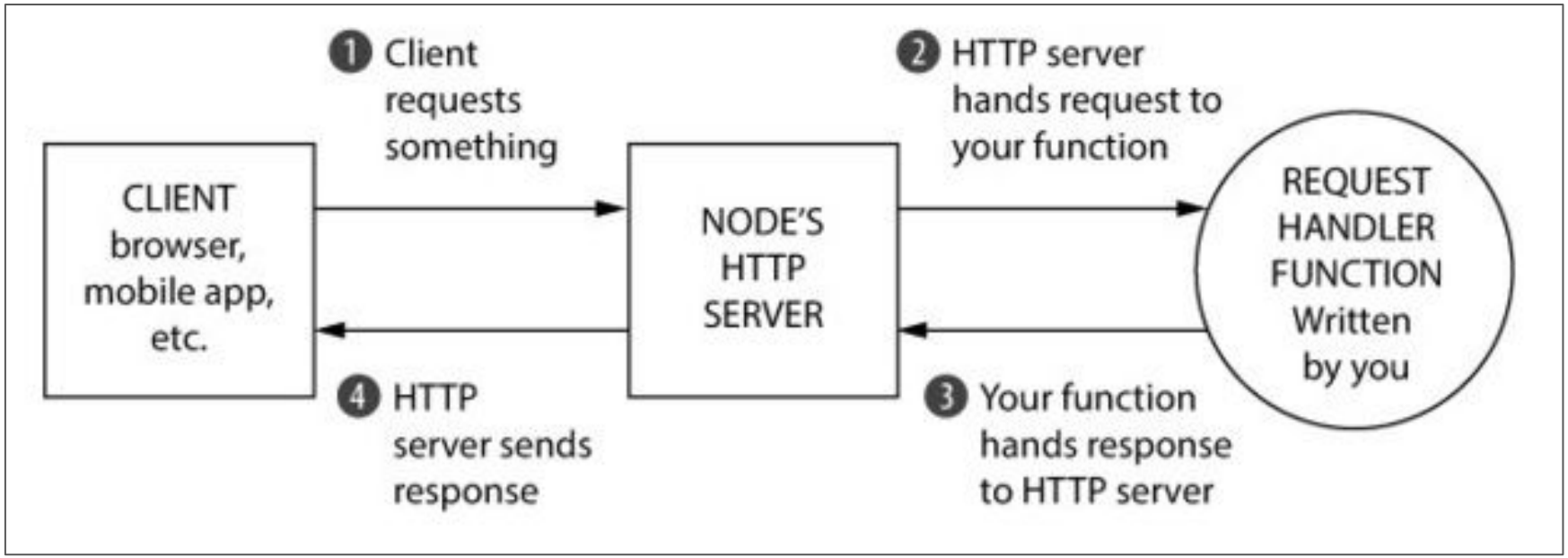
1. What platform is this running on?
2. What language is this?
3. What web framework is used?
4. What port is the app listening on?
5. When the server is opened for listening, what does the callback function do?
6. app.get() routes an HTTP GET request to the default page ('/'). What will be the response?

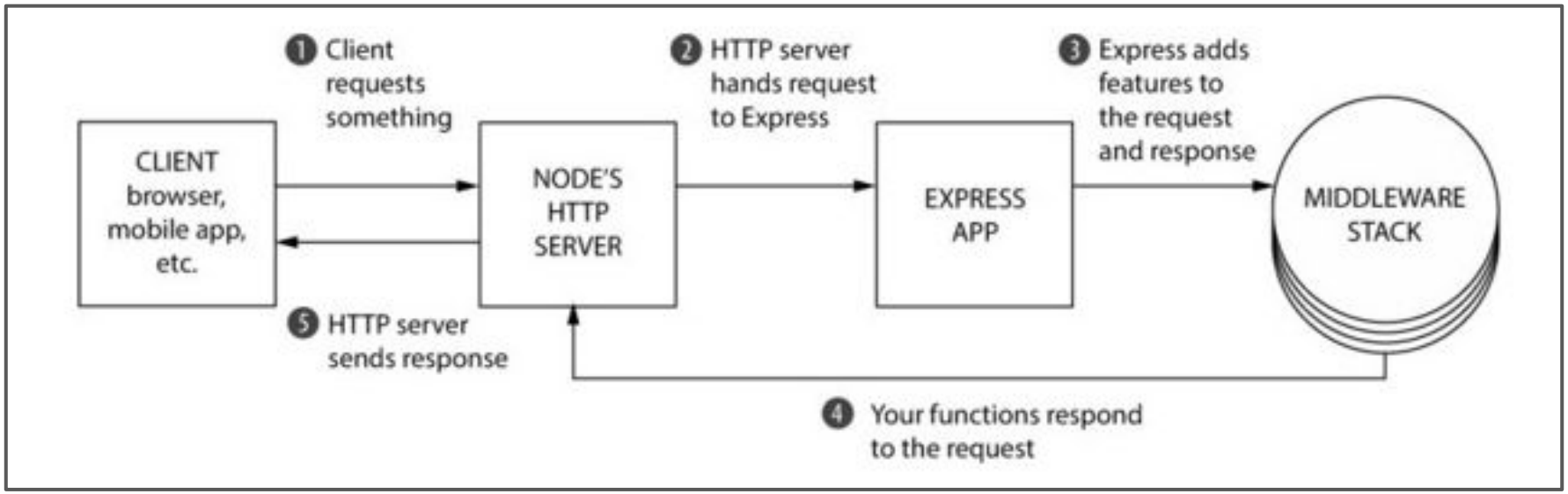
6 Reasons to ❤ Express

1. Express has built-in **routing**. In Node.js, everything gets funneled into one monolithic function, and you have to examine request.url and request.method to determine what to do. In Express, you use app.*method*(*path*,*handler*) where *method* is get, post, delete, etc., *path* is the path, and *handler* is the callback.
   a. e.g., app.**get**('/', function(req,res){ res.send("hello from root")}) // localhost:3000/
   b. app.**get**('/crossword',function(req,res){res.send("Read!")}) // localhost:3000/books/
   c. app.**put**('/genre, function(req,res){}) // put request to localhost:3000/genre
2. It has a built in mechanism for serving static files:
   a. app.use(express.static('public'))
3. Now anything in the public folder can be accessed directly, e.g.,
   a. http://localhost:3000/**images**/turtle.jpg (if public contains an **images** folder)
4. Node.js has nothing like that (it's for networks, not for the WWW).
5. Finally, Express uses **middleware**. Instead of one monolithic function, middleware consists of small functions. Each performs some task and then passes on the request, response parameters to the next one. For instance, you might have 3 middleware functions, for logging, authentication, and serving pages, called in that order.
6. Express has set a gold standard for documentation.

tldr: Express is great!

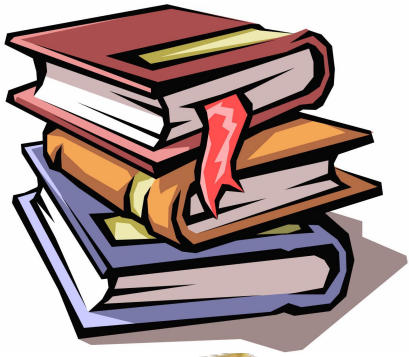
Node and Express, Visualized





Socket.IO library

- JavaScript **library** for real-time web apps e.g. a chat room
- Event-driven (like Node.js)
- Client-side library with IO, Socket, Manager
- Server-side library with Server, Socket, Client
- http://socket.io/
- API (server & client): http://socket.io/docs/



```
> npm install socket.io
```

Node-Express-Socket.io

Which runs on the server? Which runs in the browser client?

```
let express = require('express')
let app = express()

var io = require('socket.io')(app)

app.listen(80) // or http.createServer(app)

app.get('/', function (req, res) {
  res.sendfile(__dirname + '/index.html')
})

io.on('connection', function (socket) {
  socket.emit('news', { hello: 'world' })
  socket.on('my other event', function (data) {
    console.log(data)
  })
})
```

app.js

```
<script src="/socket.io/socket.io.js">
</script>
<script>

var socket =
   io.connect('http://localhost')

socket.on('news', function (data) {
    console.log(data)
    socket.emit('my other event', {
      my: 'data' })
})


</script>
```

index.html

Node-Express-Socket.io

Server

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

app.listen(80)

app.get('/', function (req, res) {
  res.sendfile(__dirname + '/index.html')
})

io.on('connection', function (socket) {
  socket.emit('news', { hello: 'world' })
  socket.on('my other event', function (data) {
    console.log(data)
  })
})
```

app.js

Client

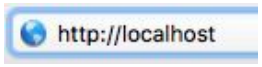
```
<script src="/socket.io/socket.io.js">
</script>
<script>

var socket =
   io.connect('http://localhost')

socket.on('news', function (data) {
    console.log(data)
    socket.emit('my other event', {
      my: 'data' })
})
```

index.html

What events are we binding?

Where does the news event get emitted?

Flow of Control



Webpage requested at some later date

Server executes first, invoking app.get() and io.on()

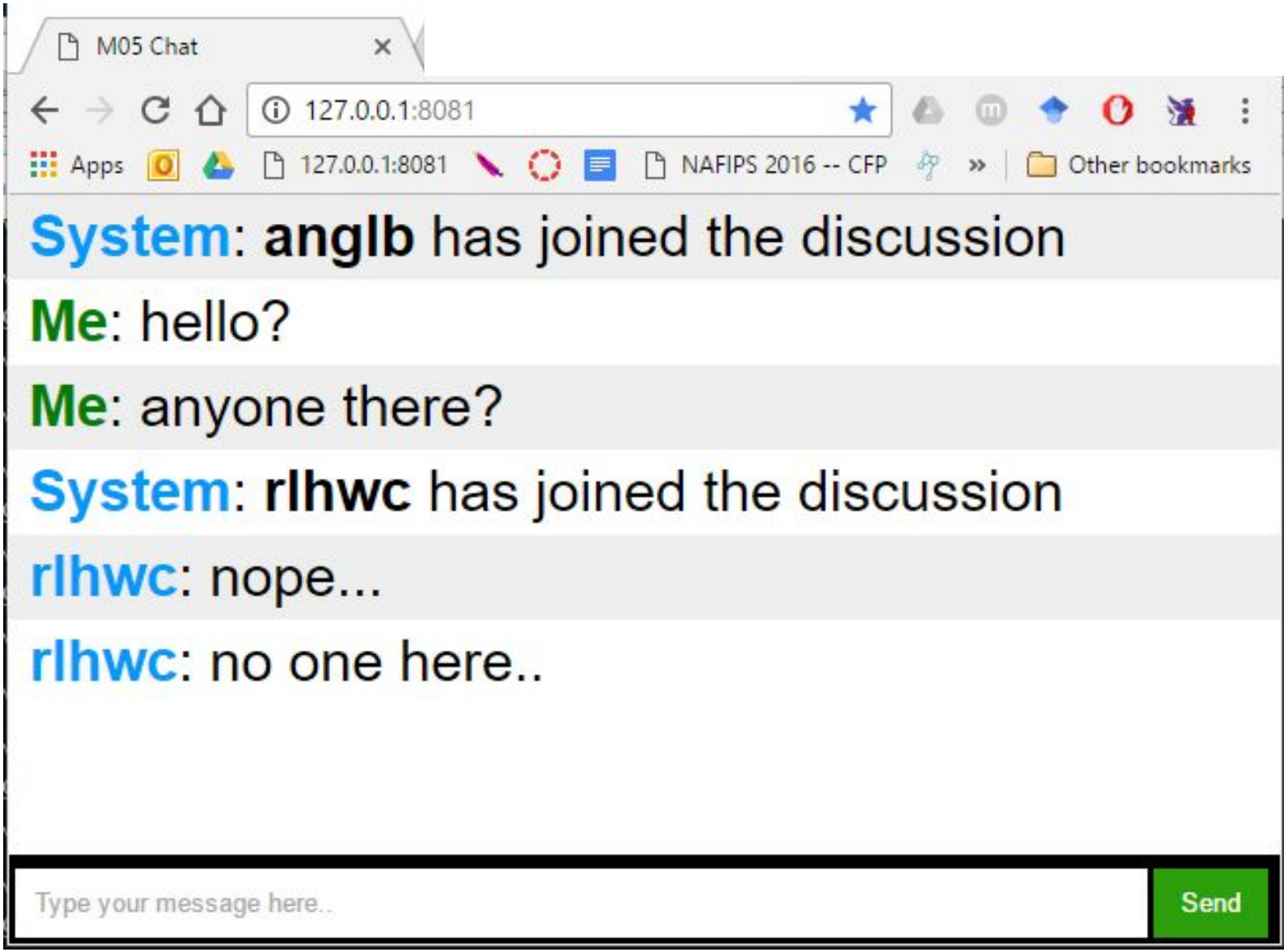
```
var app = require('express').createServer()
var io = require('socket.io')(app)

app.listen(80)
// for expository purposes, we've named our
anonymous functions as a() and b()
app.get('/', function a(req, res) {
  res.sendfile(__dirname + '/index.html')
})

io.on('connection', function b(socket) {
  socket.emit('news', { hello: 'world' })
  socket.on('my other event', function (data) {
    console.log(data)
  })
})
```

Page Delivered

```
<script src="/socket.io/socket.io.js">
</script>
<script>

var socket =
   io.connect('http://localhost')

socket.on('news', function (data) {
    console.log(data)
    socket.emit('my other event', {
      my: 'data' })
})


</script>
```

app.get(), and io.on() are called when the server starts.
a() and b() are **not**: they are registered as callbacks.
a() is called when the user connects for the first time, requesting the web page.
That causes io.connect() to be executed, which triggers b().

Chat makes a good WebSocket Example


M05 Chat
127.0.0.1:8081
Apps
127.0.0.1:8081
NAFIPS 2016 -- CFP
Other bookmarks
System: anglb has joined the discussion
Me: hello?
Me: anyone there?
System: rlhwc has joined the discussion
rlhwc: nope...
rlhwc: no one here..
Type your message here..
Send

Let's Design It

What should we use:

- For our server **platform**?
- For our web app **framework**?
- Helpful library for real-time, two-way communications?

Let's Design It

Create for **every** project:

- ❑ **.gitignore** (open Git Bash and *touch .gitignore* to create the file)
- ❑ **README.md**

Let's Design It

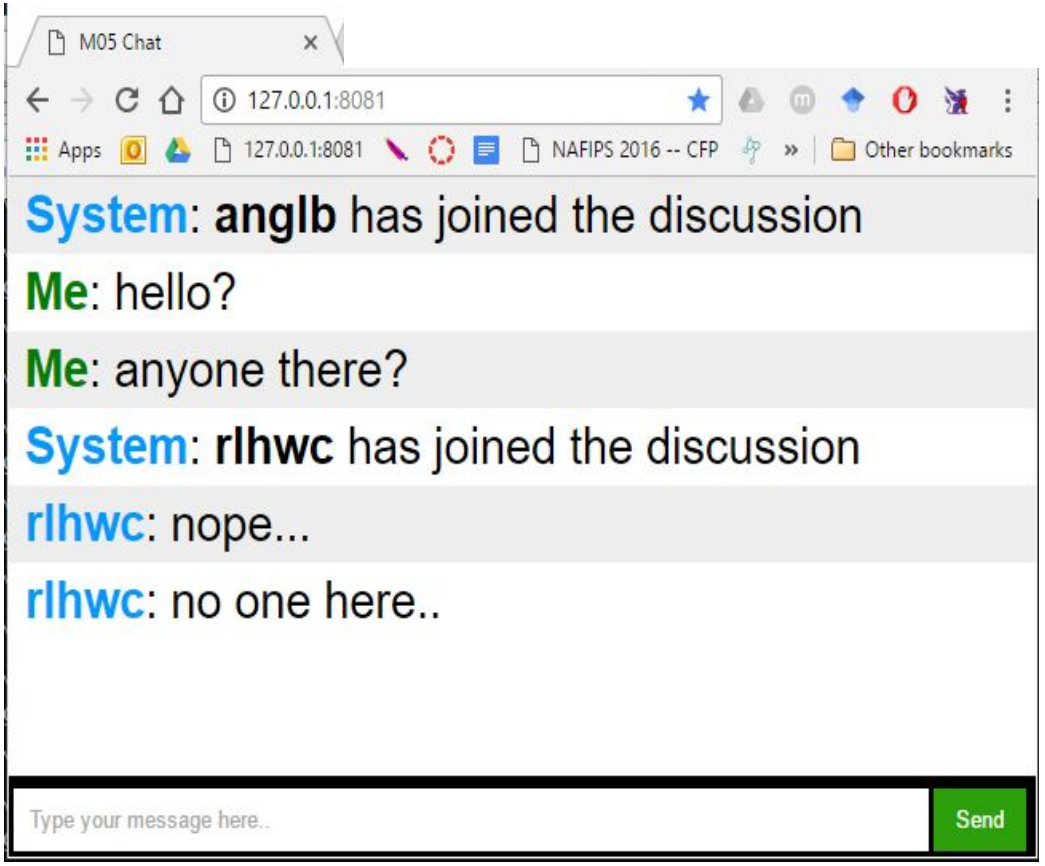
For the server:

- **package.json** (dependencies & meta information)
- **server.js** (server-side app)

Let's Design It

For the client:

- ❑ **index.html**
- ❑ **chat.css**
- ❑ **chat.js**



Let's Build It (M06)

Create C:\44563\m06 folder. In this root folder,

1. Create empty **.gitignore** (in git bash, use touch .gitignore)
2. Create empty **README.md**.
3. Create empty **package.json**.
4. Create empty **server.js**.
5. Create an **assets** subfolder. In here, create:
6. Empty **index.html**
7. Empty **chat.css**
8. Empty **chat.js**

touch is useful for creating multiple files without any content. Just write touch followed by the file names (space delimited). e.g.,
touch .gitignore README.me
would create both files. List as many as you need!
In Windows, create .gitignore. (with a dot at the end)

Let's talk through it before we begin.
How many files will we need?

.gitignore

```
node_modules
```

We'll add one line to our .gitignore so we don't commit all the code npm installs.
Why do we not commit our dependencies?

README.md

````
# M06 Chat Example

A simple chat demo using node.js, express, and socket.io

## How to use

Open a command window in your c:\44563\m06 folder.

Run npm install to install all the dependencies in the package.json file.

Run node server.js to start the server.  (Hit CTRL-C to stop.)

```
> npm install
> node server.js
```

Point your browser to `http://localhost:8081`.
````

Create a simple README.md

- what is it?
- how do you use it?

package.json

```
{
    "name": "m06",
    "version": "0.0.1",
    "description": "simple chat app",
    "main": "server.js",
    "dependencies": {
        "express": "latest",
        "socket.io": "latest"
    },
    "author": "Denise Case",
    "homepage": "https://bitbucket.org/professorcase/m
    "repository": {
        "type": "git",
        "url": "https://bitbucket.org/professorcase/m06"
    },
     "license": "Apache-2.0"
}
```

- What is the main server program?
- What modules are included?
- You'll customize this file to reflect your information.
- If lazy, like Dr. Rogers, you can use:

  npm init

server.js

.Server() is the same as .createServer()
Our express **app** is our requestListener callback

```
var express = require('express')
var app = express() // function handler
//var http = require('http').createServer(app)  // http server

// Initialize app with route / (the root) "on getting a request to /, do the following"
app.get('/', function (req, res) {
  res.write('This is where we will show our chat client.\n')
  res.write('We need an element to hold messages.\n')
  res.write('We need an element to hold notification (that others are typing).\n')
  res.write('We need an element to hold the message input form.\n')
  res.end()
})

// Listen for an application request on port 8081
// use http listen, so we can provide a callback when listening begins
// use the callback to tell the user where to point their browser
app.listen(8081, function () {
  console.log('listening on http://127.0.0.1:8081/')
})
```

http://javabeginnerstutorial.com/javascript-2/create-simple-chat-application-using-node-js-express-js-socket-io/

Let's Build It (M06)

C:\44563\m06 folder

- Created **.gitignore**
- Created **README.md**.
- Created **package.json**.
- Created **server.js**.
- Open Git Bash (or cmd window) in your folder and run:

```
> npm install
> node server.js
```

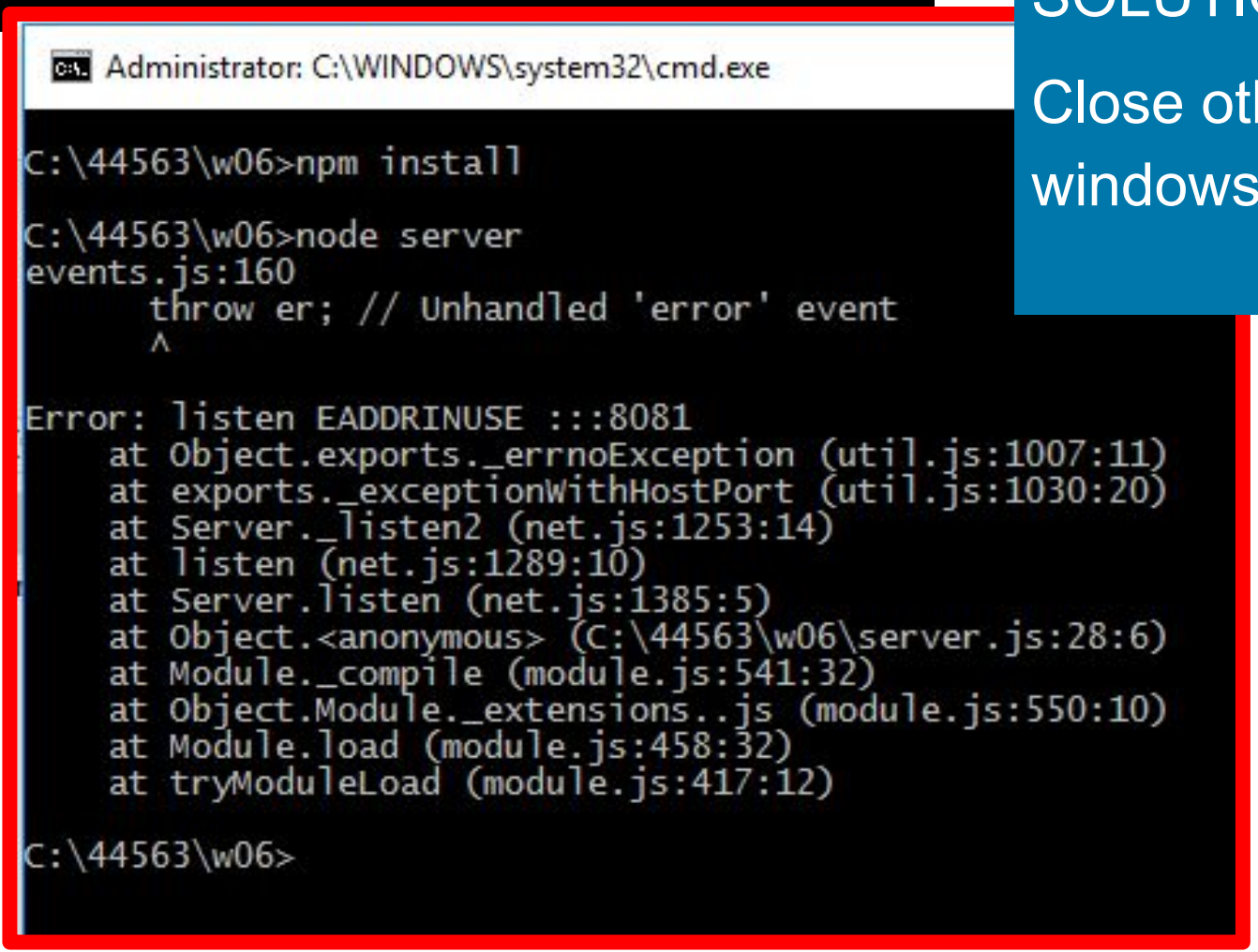
Open a browser to the path displayed

After we've built these,

1. What will npm install do?
2. Where does it get its information?
3. What does node server.js do?

Error

```
> npm install
> node server.js
```

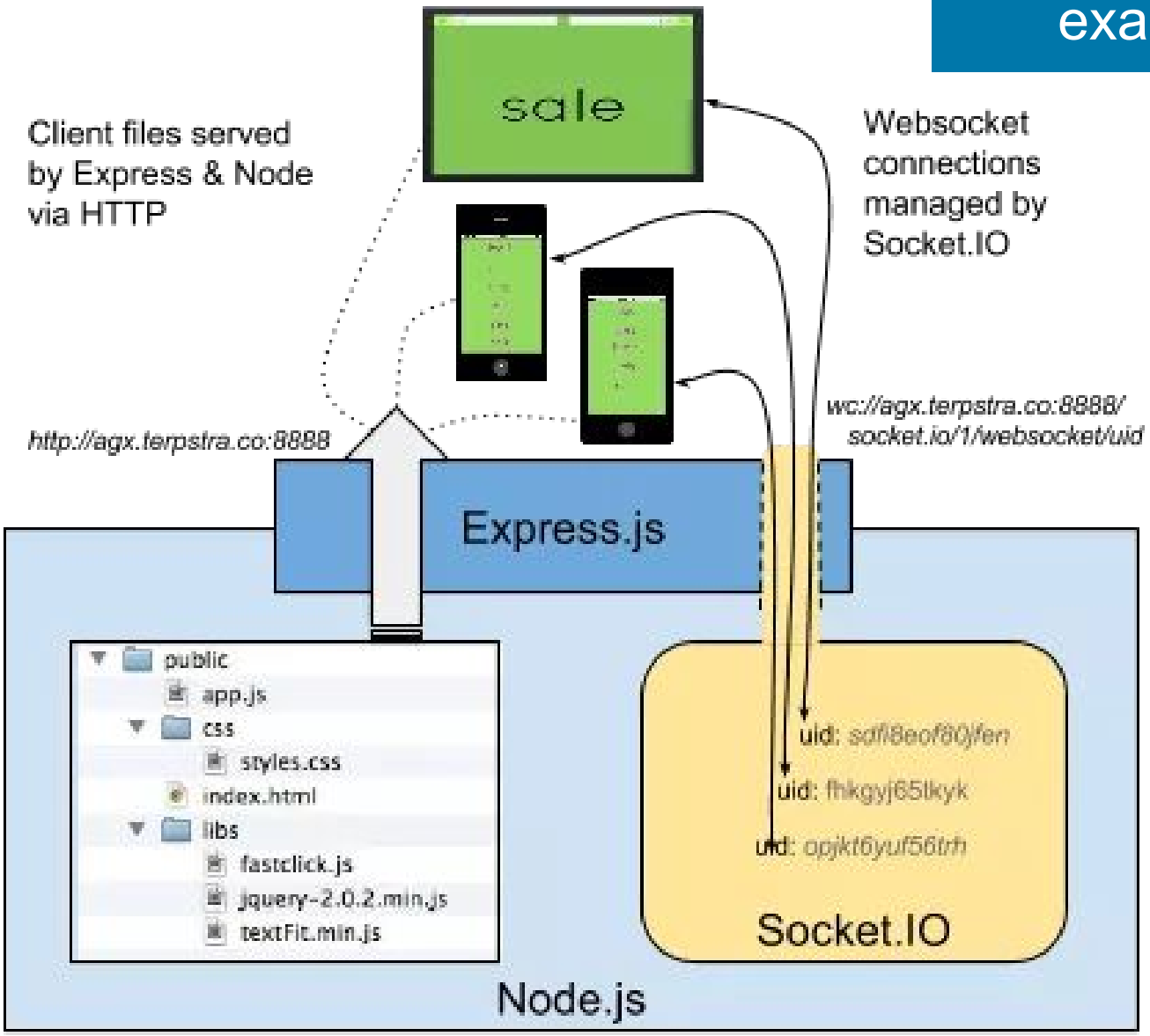


If you get **events.js:160** error, you already have a service running on that port.

SOLUTION:

Close other command windows & retry.

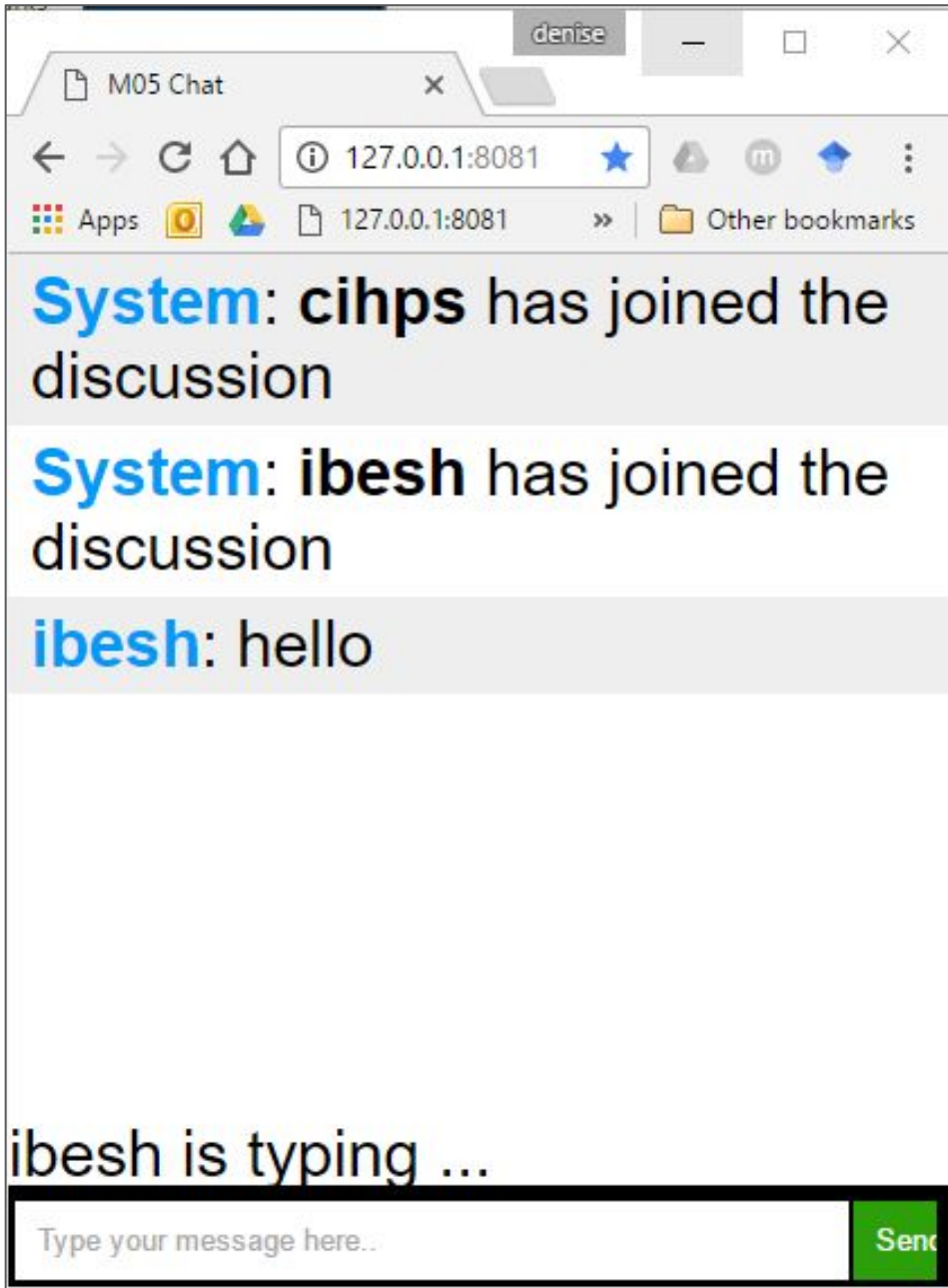
Architecture example



https://modernweb.com/2013/09/30/building-multiplayer-games-with-node-js-and-socket-io/#prettyPhoto

Adding Socket.io & finishing our chat app

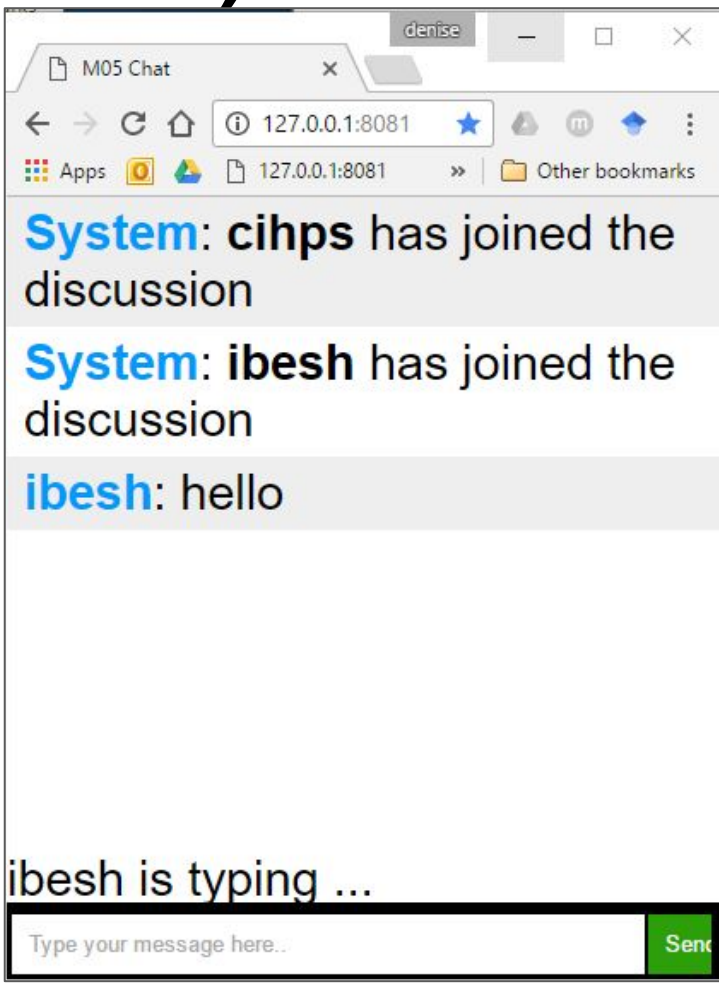
Our goal


denise
M05 Chat
127.0.0.1:8081
Apps
127.0.0.1:8081
Other bookmarks
System: cihps has joined the discussion
System: ibesh has joined the discussion
ibesh: hello
ibesh is typing ...
Type your message here..
Send

Let's Finish It (w06)

Create C:\44563**w06** folder.

1. Copy your **.gitignore**
2. Copy and update your **README.md** (now w06).
3. Copy and update your **package.json** (now w06).
4. Copy your **server.js**.
5. Copy your **assets** subfolder with:
6. Empty **index.html**
7. Empty **chat.css**
8. Empty **chat.js**



We'll start with the client-side html

assets\index.html

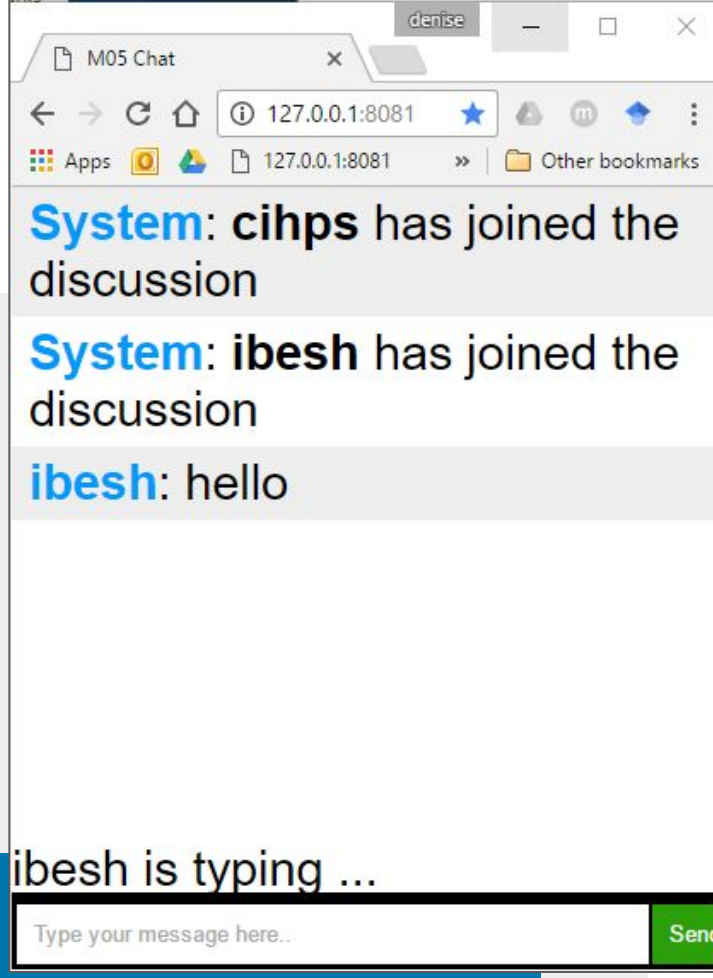
```
<!doctype html>
<html>

<head>
  <title>W05 Chat</title>
  <link rel="stylesheet" href="assets/chat.css" type="text/css" />
  <script src="https://cdn.socket.io/socket.io-1.4.5.js"></script>
  <script src="https://code.jquery.com/jquery-3.1.1.slim.min.js"></script>
  <script src="assets/chat.js"></script>
</head>

<body>


</body>

</html>
```

1. Link to css
2. Add scripts for socket.io, jquery, and app code.
3. We'll complete the body next.

http://javabeginnerstutorial.com/javascript-2/create-simple-chat-application-using-node-js-express-js-socket-io/

index.html

```
<body>



</body>
```



In the body, create 3 sections:

1. Add a <ul> for the **#messages**.
2. Add a to **#notifyU**ser.
3. Add a **#form** to input messages.

index.html

1
```
<ul id="messages"></ul>
```

2
```
<div id="notifyUser"></div>
```

3
```
<form id="form" action="" onsubmit="return submitfunction()">
    <input type="hidden" id="user" value="" />
    <input id="m" autocomplete="off" onkeyup="notifyTyping()" placeholder="Type your message here.."/><input type="submit" id="button" value="Send" />
</form>
```

Tech Notes: Use **return** in the onsubmit handler because we don't want to simply execute the function - we need to get and use the return value. Since false is returned, the submit routine is cancelled (which is fine, the script will have done what it needs to do).

In the form,

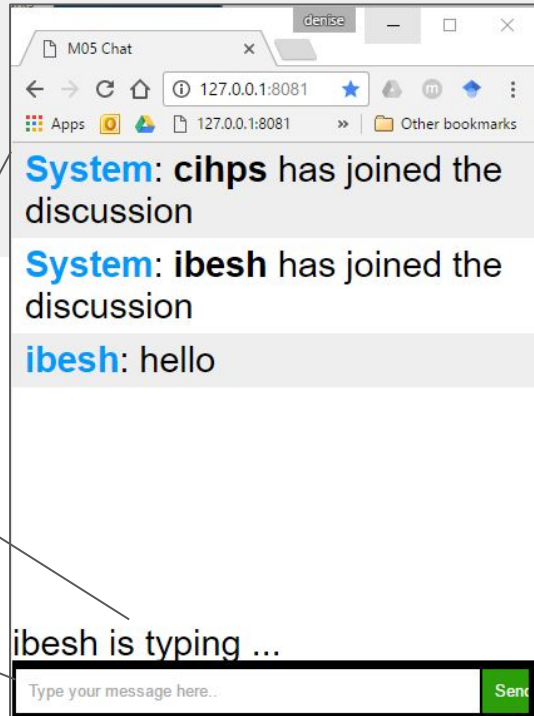
1. Assign an event handler (we want the return value).
2. Add a hidden input to hold the username.
3. Add an input text so that when a key lifts up, it calls a notify method.
4. Add a submit button "Send".

chat.css

```
* { margin: 0; padding: 0; box-sizing: border-box; }
body { font: 1.8em Helvetica, Arial; }
form { background: #000; padding: 3px; position: fixed; bottom: 0; width: 100%; }
form input { border: 0; padding: 10px; width: 90%; margin-right: .5%; }
form #button { color:#FFF; background: #2D9F0B; border: none; padding: 10px;  width: 9%; }
#messages { list-style-type: none; margin: 0; padding: 0; }
#messages li { padding: 5px 10px; }
#messages li:nth-child(odd) { background: #eee; }
#notifyUser { position: fixed; bottom: 42px; width: 100%; }
```

1. Style all.
2. Style body.
3. Style messages displaying at the top.
4. Style the notification.
5. Style the input form along the bottom.

http://javabeginnerstutorial.com/javascript-2/create-simp... at-application-using-node-js-express-js-socket-io/



chat.js: designing events

What **events** do we care about?

1. We care when a user sends a message - let's call it a **chatMessage** event.
2. We care when a user is typing - let's call each key up a **notifyUser** *(that another user is typing)* event.
3. We need to execute code each time a new client opens - when it does, we'll send a message from the system that a new user has joined. We need the code that runs on startup to emit a **special chatMessage** event.

chat.js: designing events

The client needs to:

1. Raise or "**emit**" our events.
2. **React to** each event - we can do this by associating the event name with a callback function that runs on this event.

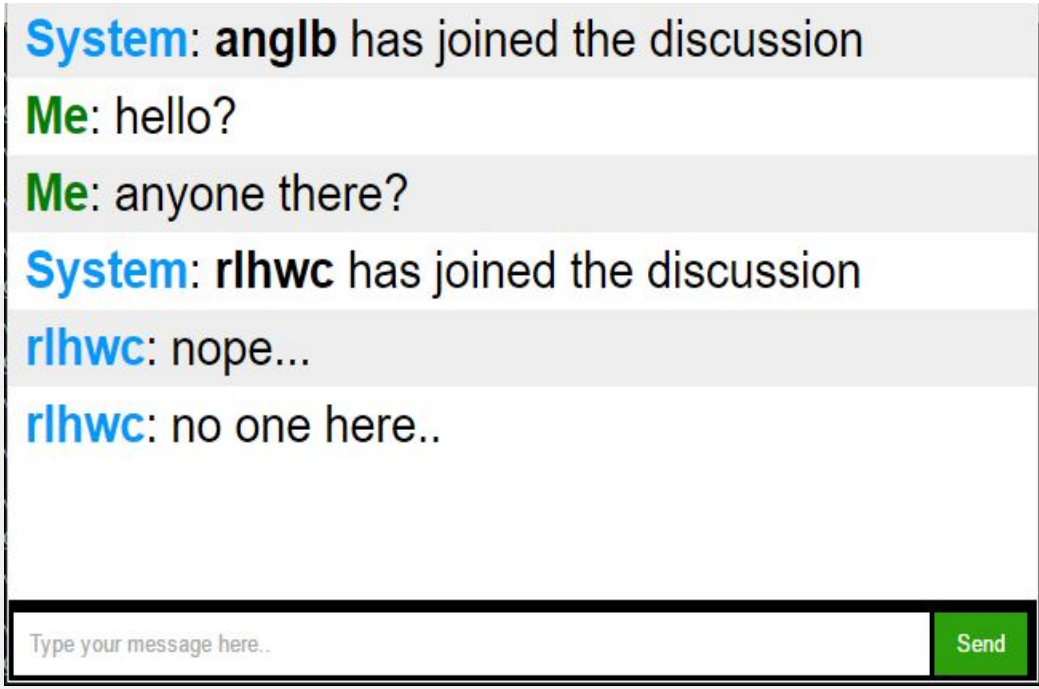
We can name our callbacks - or write them inline as anonymous functions.)

In the html, we call *return submitfunction()* and *notifyTyping()* - let's start there.

chat.js 1

```
var socket = io()  //use our helpful socket.io library to create an event emitter

// emit a new chatMessage event from the client.....

function submitfunction() {
  let from = $('#user').val()
  let message = $('#m').val()
  if (message != '') {
    socket.emit('chatMessage', from, message)// can handle more than 1 item ...
  }
  // what language and selector is used below?
  // set the value to an empty string and
  // focus on the message box again
  // return false so the form cancels the submit before sending to the server
  $('#m').val('').focus()
  return false
}
```

chat.js 2




```
// emit a new notifyUser event .....

function notifyTyping() {
  var user = $('#user').val()
  socket.emit('notifyUser', user)
}
```

```
// react to a chatMessage event.....................
// Think: on getting a chatMessage event, do this (add an <li> to our msg list)

socket.on('chatMessage', function (from, msg) {
  var me = $('#user').val()
  var color = (from == me) ? 'green' : '#009afd'
  var from = (from == me) ? 'Me' : from
  $('#messages').append('<li><b style="color:' + color + '">' + from + '</b>: ' + msg +
'</li>')
})
```

chat.js 3

1. If someone is typing (on each key up), emit a notifyUser.
2. If the user isn't me - what do we want to see? Should the notification persist forever?
3. If the user typing is me - what do we want to see?

```
// react to a notifyUser event.....
// on notifyUser, do this
socket.on('notifyUser', function (user) {
  var me = $('#user').val()
  if (user != me) {
    $('#notifyUser').text(user + ' is typing ...')
  }
  // 10 seconds after typing stops, set the notify text to an empty string
  setTimeout(function () { $('#notifyUser').text('') }, 10000)
})

// when does the document.ready() function get executed?.....
$(document).ready(function () {
  var name = makeid()
  $('#user').val(name)
  // emit a chatMessage event from the System along with a message
  socket.emit('chatMessage', 'System', '<b>' + name + '</b> has joined the discussion')
})
```

chat.js 4

```
// utility function to create a new random user name.....

function makeid() {
  var text = ""
  var possible = "abcdeghijklmnoprstuwxy"
  for (var i = 0; i < 5; i++) {
    text += possible.charAt(Math.floor(Math.random() * possible.length))
  }
  return text
}
```

One last utility function: How can we **generate a random name** for each new user?

server.js

Our previous server.js

```
var express = require('express')
var app = express()
var http = require('http').createServer(app)

// on getting a request to /, do the following
app.get('/', function (req, res) {
  res.write('This is where we will show our chat client.\n')
  res.write('We need an element to hold messages.\n')
  res.write('We need an element to hold notification (that others are typing).\n')
  res.write('We need an element to hold the message input form.\n')
  res.end()
})

// Listen for an application request on port 8081
http.listen(8081, function () {
  console.log('listening on http://127.0.0.1:8081/')
})
```

http://javabeginnerstutorial.com/javascript-2/create-simple-chat-application-using-node-js-express-js-socket-io/

server.js

Modify server.js to:

1. send back html
2. create the socket

```
// 1. add two more requires at the end of the requires
var io = require('socket.io')(http)
var path = require('path')

// 2. replace the inside lines of your app.get
app.get('/', function (req, res) {
    app.use(express.static(path.join(__dirname)))
    res.sendFile(path.join(__dirname, '../w06/assets', 'index.html'))
})

// 3. Add the following function
io.on('connection', function(socket){
     socket.on('chatMessage', function(from, msg){
         io.emit('chatMessage', from, msg)
     })
     socket.on('notifyUser', function(user){
         io.emit('notifyUser', user)
     })
})
```

Open cmd window in your project folder:

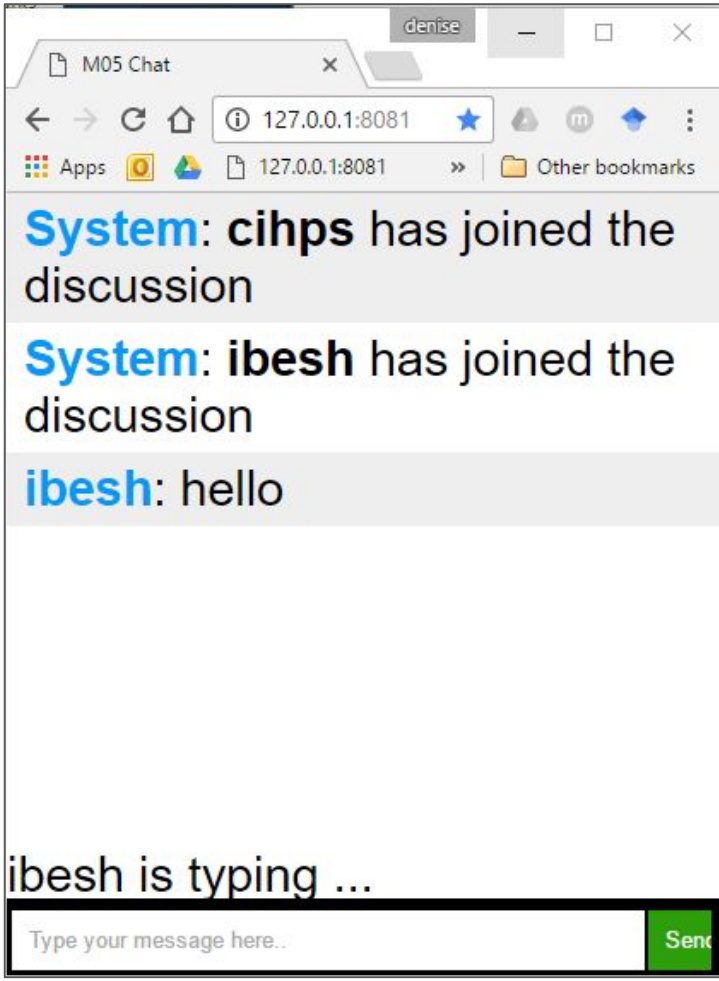
> npm install

> node server.js

http://javabeginnerstutorial.com/javascript-2/create-simple-chat-application-using-node-js-express-js-socket-io/

Finish/customize w06

1. Finish and execute your chat application.
2. As always, you are encouraged to **customize** it - use your own event names, method names, styling, etc.
3. Share a screenshot of it running on your desktop.



Finish and run your chat application

Friday

Friday - A03 begins

1. See your Assignment A03 groups.
2. Sit with your groups. Your instructor can suggest an area where each group should sit.
3. Introduce yourselves to your group.
4. Read through the assignment, following the instructions. Show off your A02. Discuss how it went.
5. Discuss expected problems as you convert your client-side web site to a hosted version running on node. (We will cover the guestbook functionality next week.)
6. Are your tools ready? Has anyone used Initializr previously?

Friday - Quiz

- Today's quiz is a bit harder.
- Take the quiz for the first time with your group.
- Take it up to three times.