Topic 14: Web Sockets

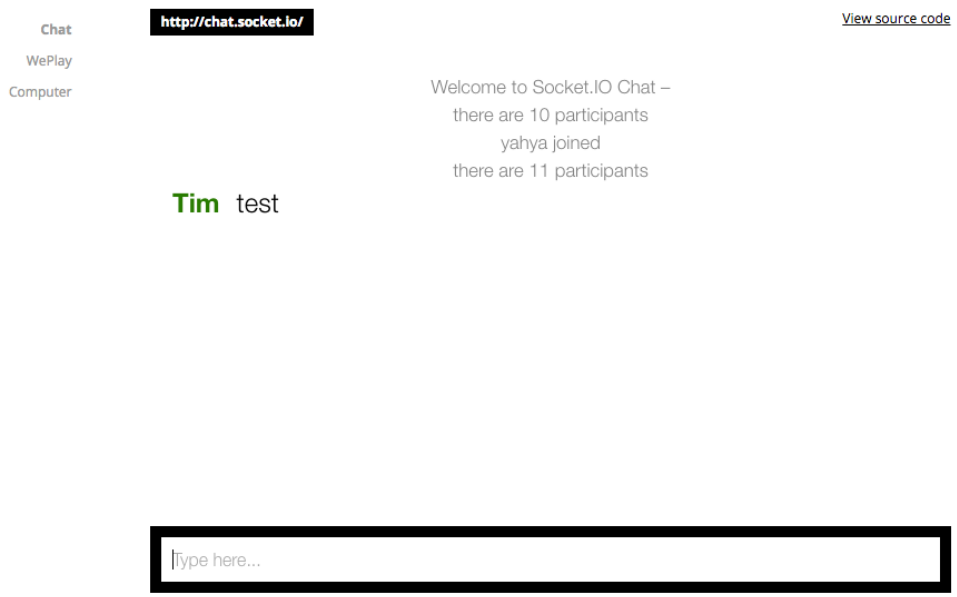
CITS3403 Agile Web Development

Getting MEAN with Mongo, Express, Angular and Node, Chapter 7

Semester 1, 2018

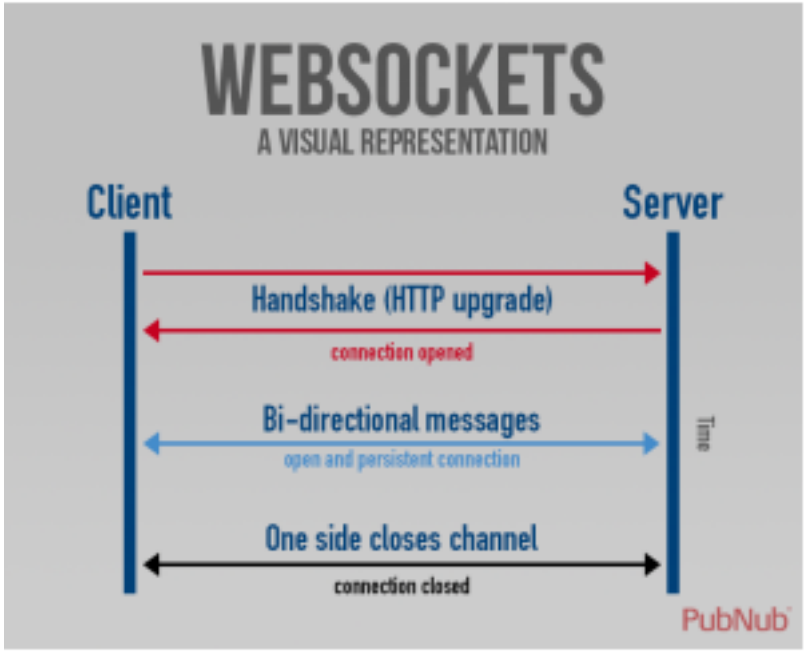
Setting up a complete project

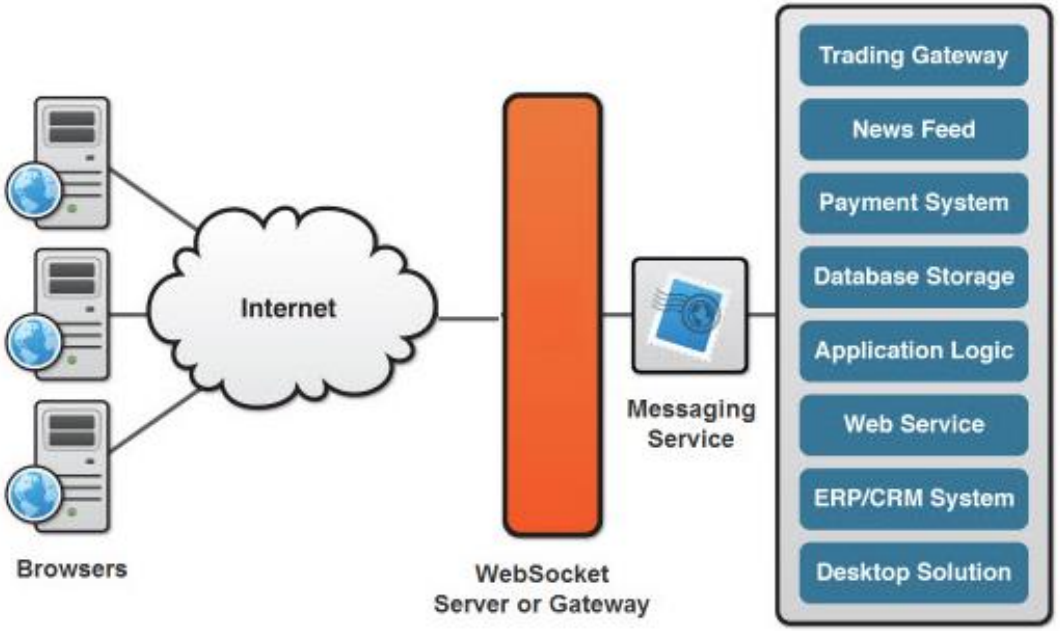
- In this lecture we will look at incorporating extra features into our application:
- First we look at real time events using web sockets.
- Sockets allow live updating of pages, like chats:



Websockets

- WebSockets allow your client-side JavaScript to open a persistent connection (stream) to the server.
- This allows real time communication in the application without having to send HTTP requests.





SocketIO

- Websockets are supported in Node via the package socketIO (see http://socket.io/)
- SocketIO is good for message passing chat or distributed games.
- For direct video and audio, WebRTC can be used.
- Clients can connect to a socket on a server, and then the server can push messages to clients.
- The client has a *listener* architecture so it will respond to the push immediately.

Sockets in an Express Project

- Sockets need to be added to the project via npm
- `npm install -save socket.io`
- socket.io uses the http server to listen for messages.
- There are various ways to include sockets, but we'll combine it with our model view architecture.
- To simplify things, we'll bring the MVC and routes folder into the root:

```
ecm-csse-022:SimpleApp tim$ ls
app.js          models          public          socket.js
bin             node_modules    routes          test
controllers     package.json    server          views
ecm-csse-022:SimpleApp tim$
```

Sockets Architecture

- We create a file socket.js in the root of the project that is similar to app.js.
- While app.js sets up a server listening for requests and sending responses, socket.js maintains a set of sockets and sends messages in response to events.

```
var io = require('socket.io')();
var ctrlChat = require('./controllers/chat');

io.on('connection', function(socket){
    ctrlChat.connect(io,socket);
    socket.on('disconnect', ctrlChat.disconnect);
    socket.on('message', function(msg){ctrlChat.message(msg, io);});
});

module.exports = io;
~
```

Sockets Architecture cont.

- `io` is a socket server and each client connects through a `socket`.
- There is a controller chat.js that supplies the callback functions to respond to chat events, and a chat model that allows chats to be logged and served.
- In this project the chat view is incorporated into layout

```
var io = require('socket.io')();
var ctrlChat = require('./controllers/chat');

io.on('connection', function(socket){
    ctrlChat.connect(io,socket);
    socket.on('disconnect', ctrlChat.disconnect);
    socket.on('message', function(msg){ctrlChat.message(msg, io);});
});

module.exports = io;
```

Registering the socket server

- The socket server is attached to the http server in `bin/www` just as app is.
- They can both share the same port.
- The socket server then responds to incoming events, such as `'connection'`

```
7  var app = require('../app');
8  var debug = require('debug')('SimpleApp:server');
9  var http = require('http');
10 //add
11 var socket = require('../socket');
12
13
14 /**
15  * Get port from environment and store in Express.
16  */
17
18 var port = normalizePort(process.env.PORT || '3000');
19 app.set('port', port);
20
21 /**
22  * Create HTTP server.
23  */
24
25 var server = http.createServer(app);
26
27 /**
28  * Listen on provided port, on all network interfaces.
29  */
30
31 server.listen(port);
32 //Add
33 socket.attach(server);
34 //Add done
35 server.on('error', onError);
36 server.on('listening', onListening);
```

Chat controller

- The server delegates callback functions to a controller, `chat.js`
- This responds the connection, disconnection and message events.
- It uses a model to save and load messages.

```
1  require('../models/db');
2  var mongoose = require('mongoose');
3  var Message = mongoose.model('Message');
4
5  module.exports.connect = function(socket){
6    console.log('User Connected');
7    Message.find().sort({time:-1}).limit(10).exec(
8        function(err, messages){
9          if(err){
10           res.render('error',{
11             message:err.message,
12             error:err
13           });
14         }
15         else{
16           console.log('last 10 messages');
17           for(var i = messages.length-1; i>=0; i--){
18             socket.emit('message', messages[i].message);
19           }
20         }
21       });
22 }
23
24 module.exports.disconnect = function(){
25   console.log('User Disconnected');
26 }
27
28 module.exports.message = function(msg, io){
29   console.log('message recieved!');
30   var message = new Message({user:'user?', message:msg, time: new Date()});
31   message.save(function(err, data){
32     if(err){
33       console.log(err);
34       res.status(500);
35       res.render('error', {
36         message:err.message,
37         error: err
38       });
39     }
40     else{
41       console.log(data, 'message saved');
42     }
43   });
44   io.emit('message', msg);
45 }
```

Chat model

- The Chat model is very simple. It just creates a schema, which is loaded in the same database from `db.js`:

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

var msgSchema = new mongoose.Schema(
      {user: String,
        message: String,
        time: Date
      });

mongoose.model('Message', msgSchema, 'messages');
~
~
```

- The only modification to `db.js` is at the final line to load the chat schema.

```
31
32 require('./person.js');
33 require('./chat.js');
~
<3/SimpleApp/models/db.js
```

Chat client

- Those are all the updates required on the backend, but the front end has to now send and receive message from the server.
- Installing `socket.io` in a project places a javascript folder in the public directory, which can be loaded in the jade file.
- We can then specify functions to send and receive messages.

```
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')

    script(src="/socket.io/socket.io.js")
    script(src="http://code.jquery.com/jquery-1.11.1.js")
    script.
        console.log('loading script');
        var socket = io();
        console.log('connected');

        function send(){
          console.log('send');
          var message = document.getElementById('message');
          socket.emit('message', message.value);
          message.value = '';
        };

        socket.on('message', function(msg){
          var messages = document.getElementById('messages');
          var newText = document.createElement('li');
          newText.innerHTML = msg;
          messages.appendChild(newText);
        });

```

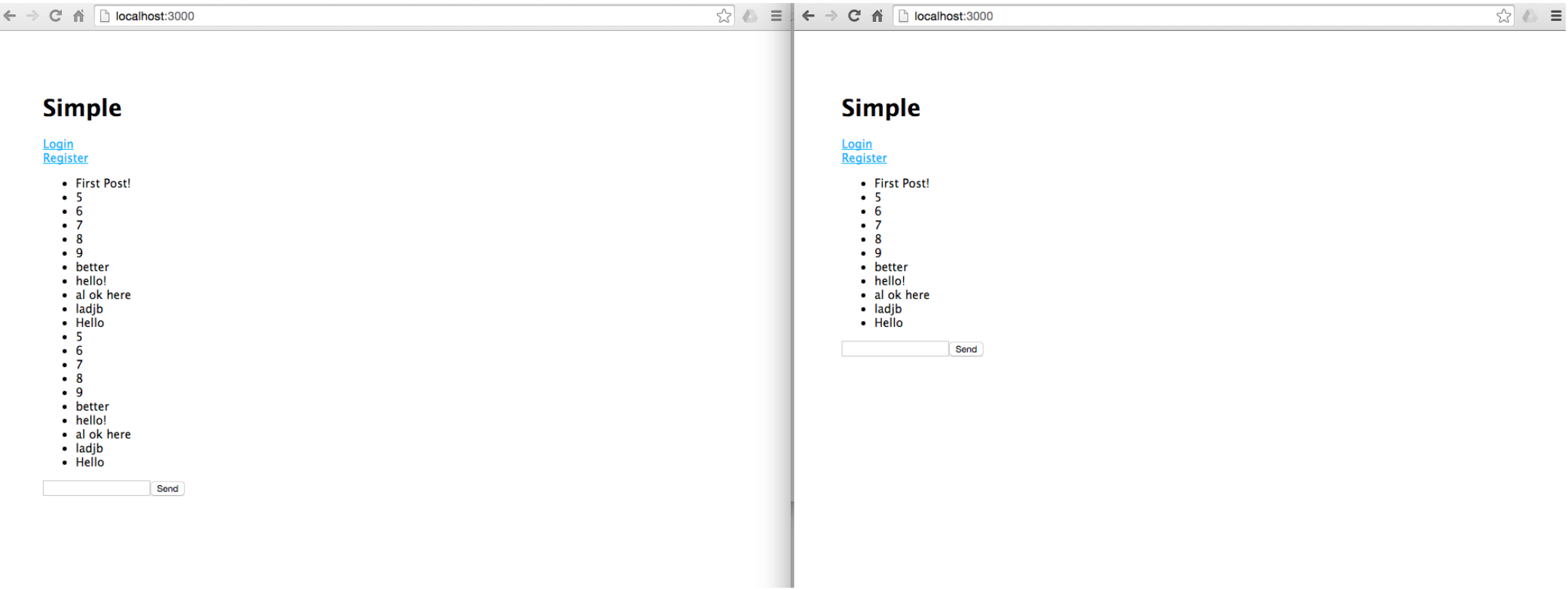
Jade file

- The chat rendering can then be done on the server side using DOM manipulation.
- This can be much neater with some Bootstrap and Angular

```
2 html
3   head
4     title= title
5     link(rel='stylesheet', href='/stylesheets/style.css')
6
7     script(src="/socket.io/socket.io.js")
8     script(src="http://code.jquery.com/jquery-1.11.1.js")
9     script.
10        console.log('loading script');
11        var socket = io();
12        console.log('connected');
13
14        function send(){
15          console.log('send');
16          var message = document.getElementById('message');
17          socket.emit('message', message.value);
18          message.value = '';
19        };
20
21        socket.on('message', function(msg){
22          var messages = document.getElementById('messages');
23          var newText = document.createElement('li');
24          newText.innerHTML = msg;
25          messages.appendChild(newText);
26        });
27
28  body
29
30    #content(width='50%', float='left')
31      block content
32
33    #chatwindow(width='20%', float='right')
34      ul(id="messages")
35        li First Post!
36
37      form(action="", id='msgForm', onSubmit='return false;')
38          input(id="message" autocomplete="off")
39          button(type='button', onClick='send();') Send
40
```

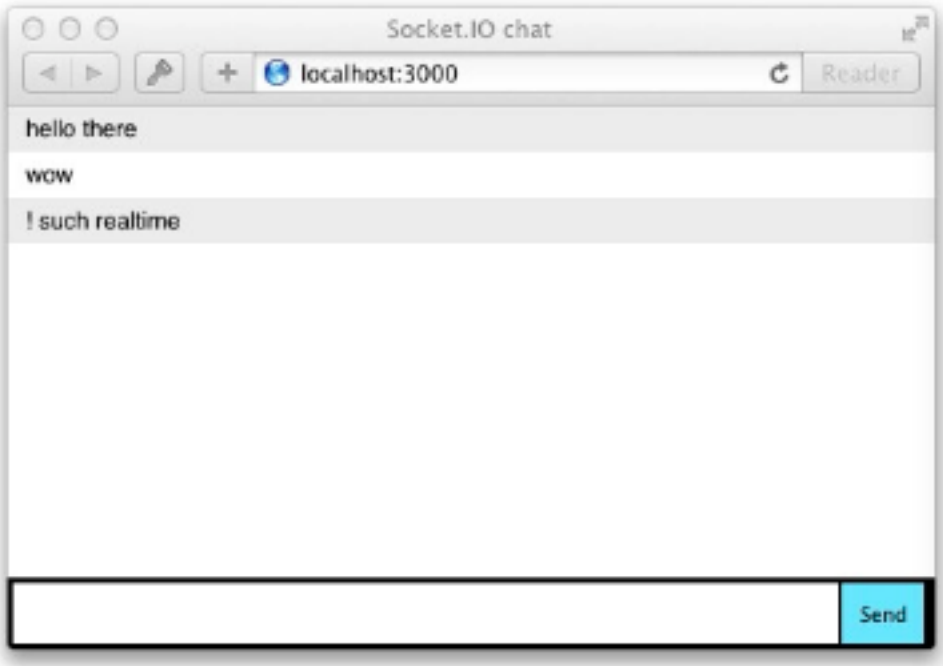
Chat output:

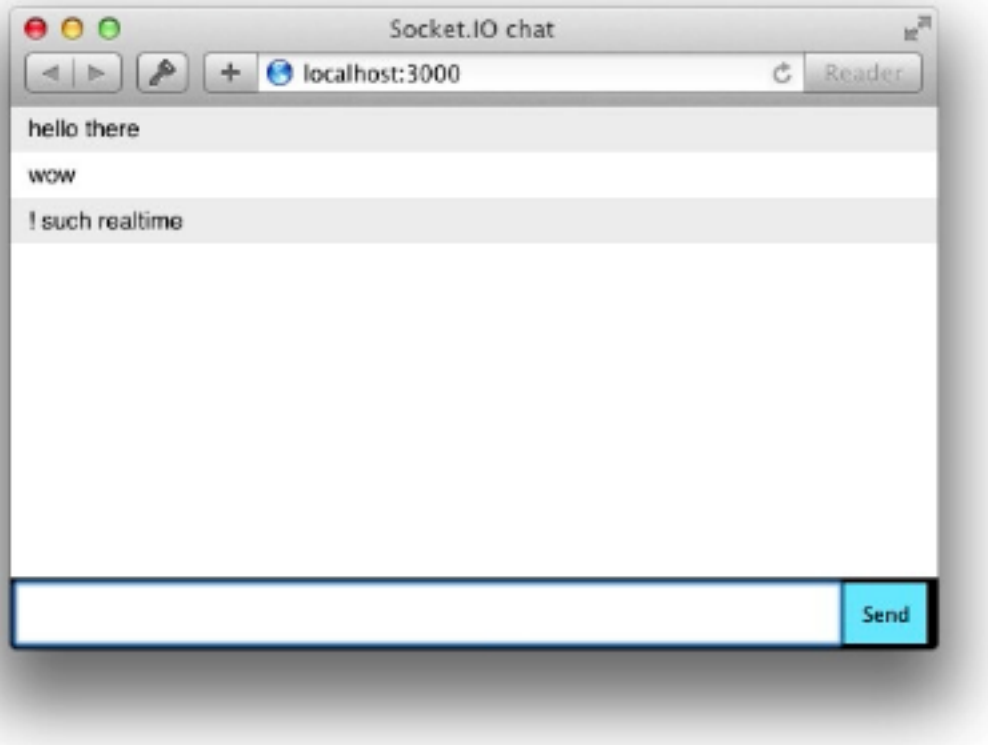
- You can now test with multiple clients sending messages and receiving messages at the same time….



Chat Window…

- This is sufficient to launch a simple chat. More details are at: http://socket.io/get-started/chat/





Other applications for sockets

- Sockets can be used for distributing real time events such as real-time scoreboards, stock prices, or weather.
- Implementing user-ids and sessions (next lecture) can allow you to have private chats between two users.
- Socket.io allows you to group sockets into namespaces and rooms, which allows you to control who can access and post messages.